

人工智能

计算Agent基础

(加) David L. Poole Alan K. Mackworth 著

董红斌 董兴业 童向荣 汪廷华 译

黄厚宽 审校

Artificial Intelligence
Foundations of Computational Agents

ARTIFICIAL INTELLIGENCE

FOUNDATIONS OF COMPUTATIONAL AGENTS



DAVID L. POOLE
ALAN K. MACKWORTH

人工智能 计算Agent基础

Artificial Intelligence Foundations of Computational Agents

“本书采用现代的思想，清晰地介绍了使用合理的计算Agent和逻辑作为统一线程的人工智能领域，包含许多完全的算例、各种难度的习题集、基于专门设计的声明式语言AIlog的编程作业、通过AIspace小程序来补充表示的整合良好的在线支持。如果你计划为本科或以上的学生教授一门人工智能的课程，必须认真对待这本非常有意思的书。”

—— 马可·瓦尔托尔塔，南卡罗来纳大学

“本书的清晰度是令人惊讶的！每一章的材料都完美地综合考虑了以下方面：针对初学者的易理解性；针对高年级学生的理论性和挑战性；针对专家的可参考性。这些材料组织成不同的章节，教师可以针对学生做出适当的分割选择。这就好像是三本书合在一起！可以肯定地说，这是21世纪AI领域必备的一本书。”

—— 杰西·霍伊，邓迪大学

“本书填补了AI文献的空白。它适用于高年级本科生，并且没有在技术严谨性上做出妥协。它是简洁的，但对AI的所有主要领域都给出了现代表示，作为AI的介绍性课程是非常有用的。Poole和Mackworth就广泛、多样的人工智能领域的条理性做了不懈的努力。在这种条理性之下，所有的AI都被放在由复杂性维度定义的智能Agent的设计空间里。”

—— 曼弗雷德·耶格，奥尔堡大学

本书讨论AI科学，它将AI作为智能计算Agent设计的研究课题。本书虽然设计为教科书，但它也适合广大专业人员和研究人员阅读。本书的一个重要特色是其在线学习资源（<http://artint.info/>）。

在过去的几十年里，人工智能是作为一种严肃科学和工程学科出现的。本书提供了针对本科生和研究生的第一手便利可用的领域综合资料，对当今该领域的基础发展进行了展望。像任何名副其实的科学一样，AI具有条理分明、形式化的理论和难以控制的实验。本书均衡了理论和实验部分，并说明了如何将理论与实验密切地联系起来，使科学与工程应用共同发展。

CAMBRIDGE
UNIVERSITY PRESS
www.cambridge.org

投稿热线：(010) 88379604
客服热线：(010) 88378991 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn



上架指导：计算机\人工智能

ISBN 978-7-111-48457-8



9 787111 484578 >

定价：79.00元

计 算 机 科 学 丛

人工智能

计算Agent基础

(加) David L. Poole Alan K. Mackworth 著

董红斌 董兴业 童向荣 汪廷华 译

黄厚宽 审校

Artificial Intelligence
Foundations of Computational Agents

ARTIFICIAL INTELLIGENCE

FOUNDATIONS OF COMPUTATIONAL AGENTS



DAVID L. POOLE
ALAN K. MACKWORTH



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

人工智能: 计算 Agent 基础 / (加) 普尔 (Poole, D. L.), (加) 麦克沃思 (Mackworth, A. K.) 著; 董红斌等译. —北京: 机械工业出版社, 2014.12

(计算机科学丛书)

书名原文: Artificial Intelligence: Foundations of Computational Agents

ISBN 978-7-111-48457-8

I. 人… II. ①普… ②麦… ③董… III. 人工智能 - 研究 IV. TP18

中国版本图书馆 CIP 数据核字 (2014) 第 258294 号

本书版权登记号: 图字: 01-2011-7728

This is a(n) simplified Chinese of the following title(s) published by Cambridge University Press:

David L. Poole and Alan K. Mackworth: Artificial Intelligence: Foundations of Computational Agents (ISBN 978-0-521-51900-7).

Original English language edition copyright © 2010 by David L. Poole and Alan K. Mackworth.

This simplified Chinese for the People's Republic of China (excluding Hong Kong, Macau and Taiwan) is published by arrangement with the Press Syndicate of the University of Cambridge, Cambridge, United Kingdom.

© Cambridge University Press and China Machine Press in 2015.

This simplified Chinese is authorized for sale in the People's Republic of China (excluding Hong Kong, Macau and Taiwan) only. Unauthorized export of this simplified Chinese is a violation of the Copyright Act. No part of this publication may be reproduced or distributed by any means, or stored in a database or retrieval system, without the prior written permission of Cambridge University Press and China Machine Press.

本书原版由剑桥大学出版社出版。

本书简体字中文版由剑桥大学出版社与机械工业出版社合作出版。未经出版者预先书面许可, 不得以任何方式复制或抄袭本书的任何部分。

此版本仅限在中华人民共和国境内 (不包括香港、澳门特别行政区及台湾地区) 销售。

本书非常全面地介绍了人工智能科学, 涵盖智能体、表达和推理、学习与规划以及个体和关系的推理等内容, 并充分考虑了 AI 科学中理论描述的形式化及实验的难以控制等特点, 对理论部分和实验部分做了很好的平衡, 展示了理论与实验之间的联系, 兼顾了科学与工程共同发展。本书提供的材料丰富、完整, 并很好地兼顾了各种层次的读者, 使之都能在自己感兴趣的书中发现感兴趣的东西。

本书适合作为计算机科学或者相关学科 (如计算机工程、哲学、认知科学和心理学) 的高年级本科生和研究生教材, 同时也适合研究人员阅读。

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 朱秀英

责任校对: 董纪丽

印 刷: 三河市宏图印务有限公司

版 次: 2015 年 1 月第 1 版第 1 次印刷

开 本: 185mm×260mm 1/16

印 张: 29.25

书 号: ISBN 978-7-111-48457-8

定 价: 79.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88378991 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzjsj@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光/邹晓东

文艺复兴以降,源远流长的科学精神和逐步形成的学术规范,使西方国家在自然科学的各个领域取得了垄断性的优势;也正是这样的传统,使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中,美国的产业界与教育界越来越紧密地结合,计算机学科中的许多泰山北斗同时身处科研和教学的最前线,由此而产生的经典科学著作,不仅擘划了研究的范畴,还揭示了学术的源变,既遵循学术规范,又自有学者个性,其价值并不会因年月的流逝而减退。

近年,在全球信息化大潮的推动下,我国的计算机产业发展迅猛,对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇,也是挑战;而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下,美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此,引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用,也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始,我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力,我们与Pearson, McGraw-Hill, Elsevier, MIT, John Wiley & Sons, Cengage等世界著名出版公司建立了良好的合作关系,从他们现有的数百种教材中甄选出Andrew S. Tanenbaum, Bjarne Stroustrup, Brian W. Kernighan, Dennis Ritchie, Jim Gray, Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, Abraham Silberschatz, William Stallings, Donald E. Knuth, John L. Hennessy, Larry L. Peterson等大师名家的一批经典作品,以“计算机科学丛书”为总称出版,供读者学习、研究及珍藏。大理石纹理的封面,也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助,国内的专家不仅提供了中肯的选题指导,还不辞劳苦地担任了翻译和审校的工作;而原书的作者也相当关注其作品在中国的传播,有的还专程为其书的中译本作序。迄今,“计算机科学丛书”已经出版了近两百个品种,这些书籍在读者中树立了良好的口碑,并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑,这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化,教育界对国外计算机教材的需求和应用都将步入一个新的阶段,我们的目标是尽善尽美,而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正,我们的联系方式如下:

华章网站: www.hzbook.com

电子邮件: hzsj@hzbook.com

联系电话: (010)88379604

联系地址: 北京市西城区百万庄南街1号

邮政编码: 100037



华章教育

华章科技图书出版中心

人工智能自 20 世纪 50 年代诞生起就受到学者们的广泛关注，到目前为止国内外很多著名高校都成立了专门研究人工智能学科的研究机构。人工智能的研究涉及多门学科的技术，因此，人工智能是一门具有挑战性的学科。伴随计算机技术的发展和全球信息化大潮的推动，人工智能的研究课题和应用领域也在不断扩展。

本书非常详尽、系统地阐述了人工智能相关技术，内容丰富、连贯，其中涵盖智能体部分、表达和推理部分、学习和规划部分以及个体和关系的推理部分等。全书共 15 章，各章相对独立却也存在一定的逻辑关系。该书在各章节中都针对不同的概念给出了示例或图形，且难度由浅入深，方便读者更加透彻地理解复杂概念或方法。本书既适合人工智能的初学者，也适合熟悉人工智能的读者，但是建议读者对启发式算法、命题与推理、概率以及算法与数据结构有一定基础之后再次阅读。

在翻译过程中，我们发现该书有以下特点：

1) 示例鲜活，书中所采用的示例都取材于实际生产、生活。该方式既阐明了技术问题，又说明了应用问题。

2) 伪代码简洁易懂，该书对算法的伪代码描述省去不必要的部分，增加适当的文字描述，算法脉络十分清晰。

3) 重点突出，该书特别强调对知识点的归纳总结，并对需要注意的问题进行提炼。

本书是一本有关 AI 的书籍，供计算机科学或者相关学科(例如计算机工程、电子工程、哲学、认知科学和心理学)的高年级本科生或研究生使用。

本书的翻译工作由董红斌、董兴业、童向荣和汪廷华共同完成，全书由黄厚宽审校。本书第 1、2、3 和 10 章由童向荣翻译，第 4、5、6、8 和 13 章由董红斌翻译，第 7、11、15 章和附录由汪廷华翻译，第 9、12 和 14 章由董兴业翻译。本书的翻译工作得到了国家自然科学基金的资助(No. 61472095)，在此表示谢意。感谢王玲玲、孙雪姣、郭艳燕、翟一鸣、侯薇、崔晓晖、滕旭阳和杨雪等老师和同学对本书翻译过程的参与和支持。

在翻译过程中，我们力求忠实、准确地把握原著并保持原著的风格。但由于水平有限和时间仓促，书中表达不准确之处在所难免，在此恳请各位专家和广大读者批评指正。

译者

2014 年 7 月

本书是一本关于人工智能(AI)科学的图书。本书认为 AI 所要研究的是如何设计智能计算 Agent(智能体)。本书采用教科书的组织形式,适合广大读者阅读。

过去几十年,我们见证了 AI 这门综合学科的兴起。正如其他学科一样, AI 具有清晰、规范的理论 and 难操控的实验部分。本书平衡了理论和实验,并将两者密切地结合。俗话说“好的理论必须有其实用价值”,因此我们将工程应用融入到 AI 的科学研究中。本书所述方法都秉承了格言“凡事都应尽量从简,但不可过简”。我们认为科学必须有其坚实的基础,因此本书介绍了基础梗概,并且提供了构建有效智能系统的相关实例,这些实例也说明了智能系统所必需的复杂事物。尽管有可能生成复杂的系统,但是其基础和构建应该是简单的。

本书可作为人工智能的教科书,适用于计算机工程、哲学、认知科学以及心理学专业的高年级本科生和研究生。本书倾向于技术层面的思考,其中包含部分技术挑战;并且本书注重实践学习:设计、构造和实现系统。每个具有科学求知欲的读者都会从本书的学习中受益。书中是根据需要来提出概念的,所以要求读者具备关于计算系统的过往经验,但在我们构造的概念上不需要具备前期基础理论的研究,诸如逻辑、概率、微积分和控制理论。

认真的学生可从本书的专业知识中获得不同层次的宝贵技能,包括规范和设计智能 Agent,在具有挑战性的应用领域中进行软件系统的实现、测试和改进。不仅如此,学生应该很高兴能够参与到智能 Agent 这门学科的兴起。能够操控现实世界中普遍存在的、智能的、嵌入的 Agent,这类实用技能也是市场的巨大需求。

智能 Agent 的研究重点在于其在环境中起到的作用。开始时我们针对简易、静态的环境来研究简单 Agent 的行为,随后逐渐增加 Agent 的能力来应对更具有挑战的环境。我们从九个方面来研究复杂性,这样可以渐进地、模块化地向读者介绍构建智能 Agent 所面临的挑战。本书结构的安排尽量使得读者可以分别理解每个方面,并且我们在四种不同的 Agent 任务中反复论证使其更加具体,这四个 Agent 任务分别是:传送机器人、诊断助手、制导系统和交易 Agent。

学生可以把 Agent 理解为一个分层设计的 Agent,该 Agent 在随机环境中具有智能行为,但是它仅能得到部分观察。该部分观察是通过个体与个体间关系的推理所得到,具有复杂的参数、行动中学习、考虑环境中的其他 Agent 和给定自身计算限制能正确处理的特点。当然我们不能一开始就设计这样的 Agent,构建此类 Agent 仍然是一个研究课题。所以我们首先介绍最简单的 Agent,进而说明如何用模块化的方法将这些复杂功能添加其中。

我们给出了几个设计选项,这些可以说明本书与同类图书的区别,包括作者早期撰写

的书籍：

- 我们给出了用于理解 AI 的清晰框架。本书不介绍那些不适合在一起研究的分散主题。例如，我们不介绍 AI 中不相关的逻辑观点和概率观点，但是却提供了一个多维设计空间，在这个空间中学生可以得到一个整体情况，并且在该空间内概率和逻辑推理可以共存。
- 我们认为最好将构成复杂技术的基础理论清晰地介绍给读者，而不是直接介绍这些复杂技术。这意味着本书所覆盖的内容与当前科学前沿有着较大的差距，但同时意味着学生可以具备较好的基础去理解当前和未来的科学研究。
- 我们难以决定的问题之一是如何线性化设计空间。在我们上一本书中 (Poole, Mackworth 和 Goebel, 1998)，较早就提出了一种关系语言，并且根据该语言构建了基础理论。但对于不具备关联的工作，该方法对于学生来说比较难以理解，比如强化学习是根据状态展开而非关系。本书采用后期关系化的方法，此方法在基于特征表达的情况下较好地反映了过去几十年研究所取得的进步。该方法使得学生可以理解概率推理和逻辑推理的互补性。虽然是采用后期关系化的方法，但是该书结构使得老师可以较早地说明各章节的关系。

本书实用的例子来自于 AIspace.org (<http://www.aispace.org>)，我们在设计过程中一直采用这些教学程序。学生如果想更多地获得构建智能系统的经验，那么也应该尝试使用 LISP 或 Prolog 这种高级的符号操作语言进行实验。我们同样提供了本书许多问题在 AILog(与 Prolog 相关的简洁的逻辑编程语言)上的实现。当然此部分对于理解和使用本书思想不是最重要的。

通过对 Agent 性能强度和表示语言这两方面的深入研究，本书方法在总结和分类各种 AI 应用上比传统方法更加简洁有效。但是例如计算视觉和计算语言学的应用在本书没有提及。

本书没有给出一个百科全书式的 AI 视图，也没去研究每一个被提出的重要思想。我们选择详尽地讲解一些基本思想，在这些基本思想上可以构造其他更复杂的技术，我们阐明了这些复杂技术是如何在此基础上被扩展的。

图 1 展示了本书所涉及的各个标题。实线给出了前提条件。前提结构通常不包含所有的子标题。为了撰写书籍，我们对各个主题进行章节编号。但是本书的讲解可以不按照章节编号，只要其满足前提条件的结构。

各章结尾处给出的参考文献并不全面：我们只是引用了我们直接使用的文献，以及那些我们认为通过引用经典工作和最近研究而提供了良好概述的文献。希望研究人员不要因为自己的研究工作被遗漏而不悦。如果您觉得书中思想有纰漏，欢迎您反馈给我们。再次重申本书不是 AI 研究的综述类图书。

诚邀您加入我们的智力探险活动：构建智能 Agent 科学。

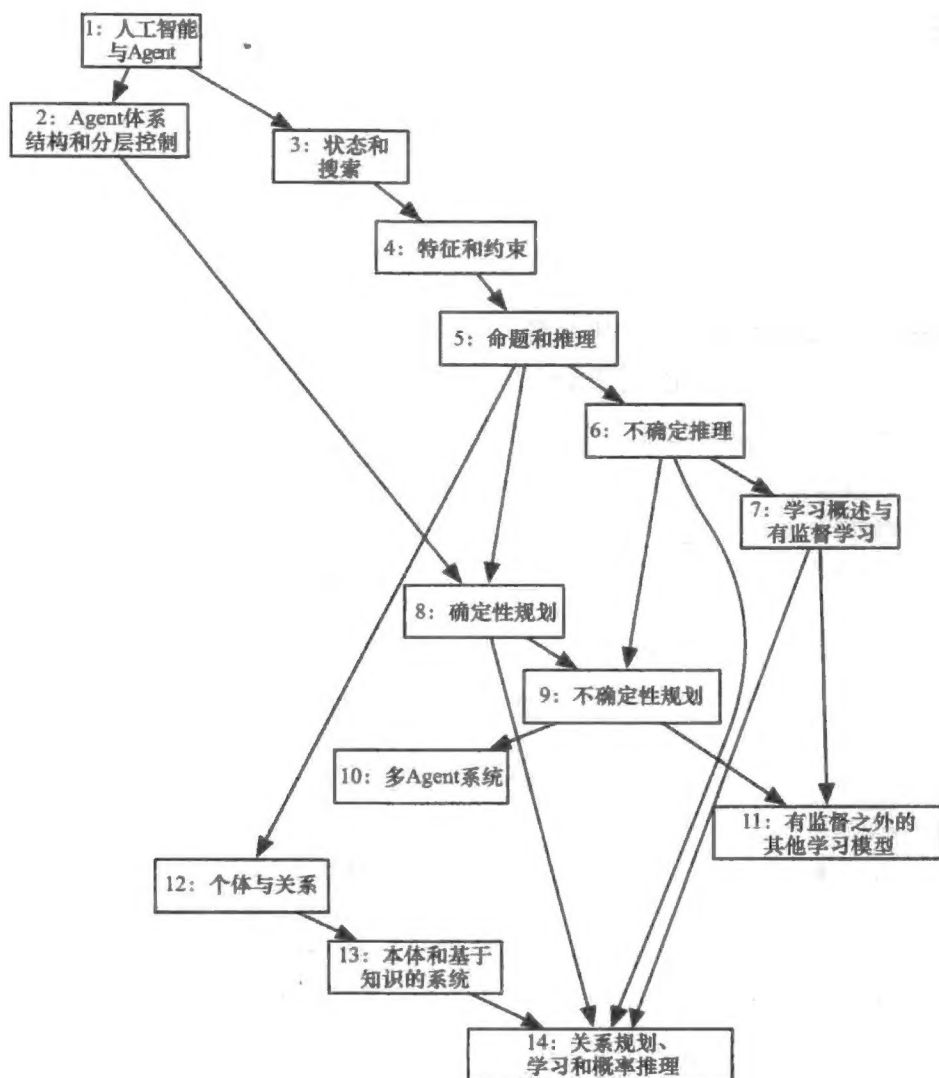


图1 章节和依赖关系的概述

致谢

感谢 Randy Goebel 对本书提出的宝贵意见。也非常感谢在早期起草本书时 Giuseppe Carenini、Cristina Conati、Mark Crowley、Pooyan Fazli、Holger Hoos、Manfred Jaeger、Mohammad Reza Khojasteh、Jacek Kiszyński、Bob Kowalski、Kevin Leyton-Brown、Marian Mackworth、Gabriel Murray、Alessandro Provetti、Marco Valtorta 和匿名审稿人提出的有益意见。感谢在草稿中指出过许多错误的学生。

感谢 Jen Fernquist 设计的网站，以及 Tom Sgouros 对 hyperlatex 所做的维护。非常感谢 James Falen 允许我们引用他的诗。感谢编辑 Lauren Cowles 和剑桥大学出版社的工作人员的支持、鼓励和帮助。如发现错误请与我们联系。

David Poole
Alan Mackworth

推荐阅读



人工智能：复杂问题求解的结构和策略（原书第6版）

作者：George F. Luger ISBN: 978-7-111-28345-4 定价：79.00元

人工智能：智能系统指南（原书第3版）

作者：Michael Negnevitsky ISBN: 978-7-111-38455-7 定价：79.00元



奇点临近

作者：Ray Kurzweil ISBN: 978-7-111-35889-3 定价：69.00元

机器学习

作者：Tom Mitchell ISBN: 978-7-111-10993-7 定价：35.00元

推荐阅读



神经网络与机器学习 (原书第3版)

作者: Simon Haykin ISBN: 978-7-111-32413-3 定价: 79.00元



机器学习导论 (原书第2版)

作者: Elthem Alpaydin ISBN: 978-7-111-45377-2 定价: 59.00元



数据挖掘: 实用机器学习工具与技术 (原书第3版)

作者: Ian H. Witten 等 ISBN: 978-7-111-45381-9 定价: 79.00元

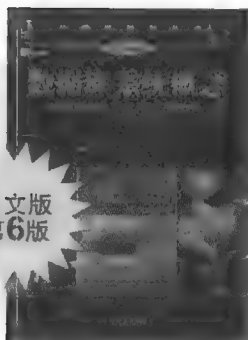


机器学习基础教程

作者: Simon Rogers 等 ISBN: 978-7-111-40702-7 定价: 45.00元

推荐阅读

中文版
第6版



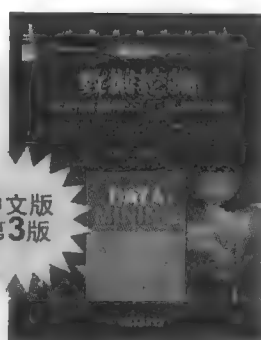
作者: Abraham Silberschatz 著
中文翻译版: 978-7-111-37529-6, 99.00元
英文精编版: 978-7-111-40086-8, 89.00元
本科教学版: 978-7-111-40085-1, 59.00元

中文版
第3版



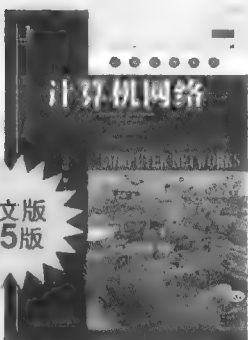
作者: Jiawei Han 等著
英文版: 978-7-111-37431-2, 118.00元
中文版: 978-7-111-39140-1, 79.00元

中文版
第3版



作者: Ian H. Witten 等著
英文版: 978-7-111-37417-6, 108.00元
中文版: 978-7-111-45381-9, 79.00元

英文版
第5版



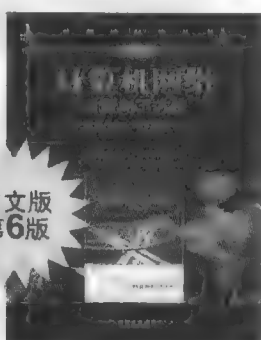
作者: Andrew S. Tanenbaum 著
书号: 978-7-111-35925-8, 99.00元

中文版



作者: Behrouz A. Forouzan 著
英文版: 978-7-111-37430-6, 79.00元
中文版: 978-7-111-40088-2, 99.00元

中文版
第6版



作者: James F. Kurose 著
书号: 978-7-111-45378-9, 79.00元

中文版
第3版

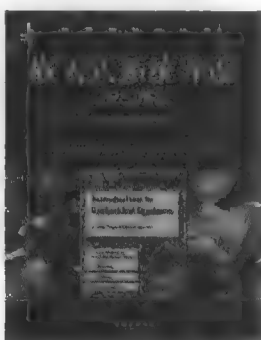


作者: Thomas H. Cormen 等著
书号: 978-7-111-40701-0, 128.00元

英文版
第5版



作者: John L. Hennessy 著
书号: 978-7-111-38458-0, 138.00元



作者: Edward Ashford Lee 著
书号: 978-7-111-36021-6, 65.00元

目 录

Artificial Intelligence: Foundations of Computational Agents

出版者的话

译者序

前 言

第一部分 世界中的 Agent：什么是 Agent 及如何创建它们

第 1 章 人工智能与 Agent	2
1.1 什么是人工智能	2
1.2 人工智能简史	4
1.3 环境中的 Agent	6
1.4 知识表示	7
1.4.1 定义解	8
1.4.2 表示	9
1.4.3 推理与行为	10
1.5 复杂性维度	12
1.5.1 模块性	12
1.5.2 表示方案	13
1.5.3 规划期	14
1.5.4 不确定性	15
1.5.5 偏好	15
1.5.6 Agent 数量	16
1.5.7 学习	16
1.5.8 计算限制	17
1.5.9 多维交互	18
1.6 原型应用	19
1.6.1 自主传递机器人	19
1.6.2 诊断助手	21
1.6.3 智能指导系统	23
1.6.4 交易 Agent	24
1.7 本书概述	25
1.8 本章小结	26
1.9 参考文献及进一步阅读	26
1.10 习题	27
第 2 章 Agent 体系结构和分层控制	28
2.1 Agent	28

2.2 Agent 系统	29
2.3 分层控制	32
2.4 嵌入式和仿真 Agent	37
2.5 通过推理来行动	38
2.5.1 设计时间与离线计算	39
2.5.2 在线计算	40
2.6 本章小结	41
2.7 参考文献及进一步阅读	42
2.8 习题	42

第二部分 表达和推理

第 3 章 状态和搜索	46
3.1 用搜索进行问题求解	46
3.2 状态空间	47
3.3 图搜索	48
3.4 一个通用搜索算法	50
3.5 无信息搜索策略	51
3.5.1 深度优先搜索	51
3.5.2 宽度优先搜索	55
3.5.3 最低花费优先搜索	56
3.6 启发式搜索	56
3.6.1 A* 搜索	58
3.6.2 搜索策略总结	59
3.7 更复杂的搜索方法	60
3.7.1 环检查	60
3.7.2 多路径剪枝	60
3.7.3 迭代深化	62
3.7.4 分支界限法	63
3.7.5 搜索方向	65
3.7.6 动态规划法	66
3.8 本章小结	68
3.9 参考文献及进一步阅读	68
3.10 习题	69
第 4 章 特征和约束	71
4.1 特征和状态	71

4.2 可能世界、变量和约束	72	5.5.2 完备知识的验证程序	130
4.2.1 约束	74	5.6 溯因推理	132
4.2.2 约束满足问题	75	5.7 因果模型	135
4.3 生成-测试算法	76	5.8 本章小结	137
4.4 使用搜索求解 CSP	76	5.9 参考文献及进一步阅读	137
4.5 一致性算法	78	5.10 习题	138
4.6 域分割	81	第6章 不确定推理	145
4.7 变量消除	83	6.1 概率	145
4.8 局部搜索	85	6.1.1 概率的语义	146
4.8.1 迭代最佳改进	86	6.1.2 概率公理	147
4.8.2 随机算法	87	6.1.3 条件概率	149
4.8.3 评估随机算法	90	6.1.4 期望值	152
4.8.4 局部搜索中利用命题结构	92	6.1.5 信息理论	153
4.9 基于种群的方法	92	6.2 独立性	153
4.10 最优化	94	6.3 信念网络	155
4.10.1 最优化的系统方法	96	6.4 概率推理	164
4.10.2 局部搜索最优化	98	6.4.1 信念网络中的变量消除	164
4.11 本章小结	99	6.4.2 通过随机模拟进行近似推理	169
4.12 参考文献及进一步阅读	100	6.5 概率和时间	176
4.13 习题	100	6.5.1 马尔可夫链	176
第5章 命题和推理	103	6.5.2 隐马尔可夫模型	176
5.1 命题	103	6.5.3 监听和平滑算法	179
5.1.1 命题演算的语法	103	6.5.4 动态信念网络	180
5.1.2 命题演算的语义	104	6.5.5 时间粒度	181
5.2 命题确定子句	107	6.6 本章小结	181
5.2.1 问题与解答	109	6.7 参考文献及进一步阅读	181
5.2.2 验证	110	6.8 习题	182
5.3 知识表示问题	115		
5.3.1 背景知识与观察	115	第三部分 学习与规划	
5.3.2 询问用户	116	第7章 学习概述与有监督学习	186
5.3.3 知识层的解释	117	7.1 学习问题	186
5.3.4 知识层的调试	119	7.2 有监督学习	189
5.4 反证法验证	122	7.2.1 评估预测	190
5.4.1 Horn 子句	122	7.2.2 无输入特征的点估计	193
5.4.2 假说与冲突	123	7.2.3 概率学习	195
5.4.3 基于一致性的诊断	124	7.3 有监督学习的基本模型	196
5.4.4 通过假设和 Horn 子句推理	126	7.3.1 决策树学习	196
5.5 完备知识假设	127	7.3.2 线性回归与分类	200
5.5.1 非单调推理	130	7.3.3 贝叶斯分类器	203

7.4 组合模型	206	9.5.3 值迭代	268
7.4.1 神经网络	207	9.5.4 策略迭代	270
7.4.2 集成学习	210	9.5.5 动态决策网络	271
7.5 避免过拟合	210	9.5.6 部分可观察决策过程	273
7.5.1 最大后验概率和最小描述 长度	211	9.6 本章小结	273
7.5.2 交叉验证	213	9.7 参考文献及进一步阅读	274
7.6 基于案例的推理	213	9.8 习题	274
7.7 改进假设空间的学习	215	第10章 多 Agent 系统	280
7.7.1 变型空间学习	216	10.1 多 Agent 框架	280
7.7.2 可能近似正确学习	218	10.2 博弈的表示	281
7.8 贝叶斯学习	220	10.2.1 博弈的标准形式	281
7.9 本章小结	224	10.2.2 博弈的扩展形式	282
7.10 参考文献及进一步阅读	225	10.2.3 多 Agent 决策网络	283
7.11 习题	225	10.3 完全信息的计算策略	284
第8章 确定性规划	229	10.4 部分可观察的多 Agent 推理	286
8.1 状态、动作以及目标的表示	229	10.4.1 纳什均衡计算	290
8.1.1 显式状态空间表示法	230	10.4.2 学习协调	292
8.1.2 基于特征的动作表示	231	10.5 群体决策	294
8.1.3 STRIPS 表示法	232	10.6 机制设计	294
8.1.4 初始状态和目标	233	10.7 本章小结	296
8.2 前向规划	233	10.8 参考文献及进一步阅读	297
8.3 回归规划	235	10.9 习题	297
8.4 CSP 规划	236	第11章 有监督之外的其他学习 模型	298
8.5 偏序规划	238	11.1 聚类	298
8.6 本章小结	241	11.1.1 期望最大化	298
8.7 参考文献及进一步阅读	241	11.1.2 k -均值	299
8.8 习题	241	11.1.3 用于软聚类的期望最大化	300
第9章 不确定性规划	244	11.2 信念网络学习	303
9.1 偏好和效用	245	11.2.1 概率学习	303
9.2 一次性的决策	252	11.2.2 未观察到的变量	304
9.3 序贯决策	255	11.2.3 缺失数据	304
9.3.1 决策网络	256	11.2.4 结构学习	305
9.3.2 策略	258	11.2.5 信念网络学习的一般情形	306
9.3.3 决策网络的变量消除	259	11.3 增强学习	306
9.4 信息与控制的价值	262	11.3.1 演化算法	308
9.5 决策过程	264	11.3.2 时间差	308
9.5.1 策略值	267	11.3.3 Q -学习	309
9.5.2 最优策略值	267	11.3.4 探索与利用	312

11.3.5 增强学习算法的评估	313	13.2 灵活的表示	363
11.3.6 在策略学习	314	13.2.1 选择个体和关系	364
11.3.7 为路径分配信用和责任	315	13.2.2 图形化表示	366
11.3.8 基于模型的方法	317	13.2.3 原始关系与导出关系	369
11.3.9 基于特征的增强学习	319	13.3 本体与知识共享	373
11.4 本章小结	320	13.3.1 描述逻辑	376
11.5 参考文献及进一步阅读	321	13.3.2 顶层本体	380
11.6 习题	321	13.4 查询用户和其他知识来源	382
第四部分 个体与关系的推理		13.4.1 函数化关系	383
第12章 个体与关系	324	13.4.2 更普遍的问题	383
12.1 在特征之外利用结构	324	13.5 实现基于知识的系统	384
12.2 符号与语义	325	13.5.1 基语言和元语言	385
12.3 Datalog: 一个关联规则语言	326	13.5.2 普通的元解释器	386
12.3.1 基 Datalog 的语义	328	13.5.3 扩展基语言	387
12.3.2 解释变量	329	13.5.4 深度有限搜索	388
12.3.3 带变量的查询	333	13.5.5 元解释器构建证明树	389
12.4 证明与替换	334	13.5.6 可询问用户的元解释器	390
12.4.1 带变量的自底向上过程	335	13.5.7 推迟目标	391
12.4.2 带变量的确定性归结	337	13.6 本章小结	391
12.5 函数符号	339	13.7 参考文献及进一步阅读	392
12.6 在自然语言处理中的应用	344	13.8 习题	392
12.6.1 在上下文无关文法中使用 限定子句	345	第14章 关系规划、学习和概率 推理	396
12.6.2 增强文法	347	14.1 规划个体与关系	396
12.6.3 为非终结符号建立结构	348	14.1.1 情景演算	396
12.6.4 封装的文本输出	348	14.1.2 事件演算	401
12.6.5 强制约束	349	14.2 个体与关系的学习	402
12.6.6 建立自然语言与数据库的 接口	350	14.3 概率关系模型	406
12.6.7 局限	351	14.4 本章小结	410
12.7 相等	352	14.5 参考文献及进一步阅读	411
12.7.1 允许相等断言	352	14.6 习题	411
12.7.2 唯一名字假设	353	第五部分 宏观图景	
12.8 完备知识假设	355	第15章 回顾与展望	416
12.9 本章小结	358	15.1 复杂性维度回顾	416
12.10 参考文献及进一步阅读	358	15.2 社会与道德后果	418
12.11 习题	359	15.3 参考文献及进一步阅读	420
第13章 本体和基于知识的系统	363	附录A 数学基础与记号	421
13.1 知识共享	363	参考文献	425
		索引	439

| 第一部分 |

Artificial Intelligence, Foundations of Computational Agents

世界中的 Agent: 什么是 Agent 及如何创建它们

人工智能与 Agent

人工智能的历史充满幻想、可能、验证和希望。自从荷马描绘机器“鼎”服侍在众神的餐桌旁，那想象中的机器佣人便成为我们文化的一部分。然而，我们人工智能的研究者，直到 50 年前，才首次制造出实验性机器来验证那些假想，即有关具备思维和智能行为机器人的假想，使得之前仅在理论上具备可能性的机器人得到验证。

——Bruce Buchanan [2005]

历经几个世纪的思想构建，人工智能学科被公认为有超过 50 年的历史，本书正是与此有关。如同以上 Buchanan 所指出的，我们现已验证这种思想假设，并用于解决实际问题。许多科学和工程问题已经得到解决，但更多其他问题亟待解决。纵然已存在人工智能的实际应用，仍有很多潜在应用等待挖掘。本书中，我们将揭示具有智能行为的计算 Agent 的基本原理，这将有助于大家理解人工智能现在和未来的工作，并有助于对这些原理做出进一步的贡献。

1.1 什么是人工智能

人工智能(AI)是一个研究能够产生智能行为的计算 Agent 的综合与分析的领域。下面阐述此定义的每个部分。

Agent 是指在某个环境中进行某种行为的个体——它完成了某件事。Agent 可以是蠕虫、狗、恒温器、飞机、机器人、人类、公司、国家中的任何一个。

我们感兴趣的是一个 Agent 做什么？怎么做？我们通过 Agent 的行为来辨别不同的 Agent。

当 Agent 具有以下行为时，我们说此 Agent 是智能的：

- Agent 的行为能够与其环境及目标相适应。
- 能够灵活地适用于改变的环境及目标。
- 能够从经验中学习。
- 给定感知和计算限制时能够做出恰当的选择。Agent 通常不能直接观察到整个世界，它只有有限的记忆，并且没有无限时间进行某个行为。

计算 Agent 可以通过一系列的计算对其行为进行决策。也就是说，决策可以分解成能够在硬件设备上实现的一系列原始操作。这种计算有很多形式，人类是通过一些软件完成的，而计算机则是由一些硬件完成的。尽管有一些 Agent 可以认为是不可计算的，例如自然现象中侵蚀景观的风和雨。但是，是否所有的 Agent 都是可计算的，仍是一个开放性问题。

人工智能的核心科学目标是理解自然系统或人工系统中一些智能行为的原理，主要通过以下几个部分来进行：

- 对自然 Agent 及人工 Agent 进行分析。
- 对如何构造 Agent 进行形式化定义，并进行验证。
- 设计、构建和实验计算系统，通过执行某些智能性任务来进行验证。

作为科学的一部分，研究者们已建立了一些实验系统来验证某些假设，或来探索可能性空间。这与适用于某个实际领域的那些应用系统有很大不同。

注意，这个定义不是关于智能思想的。我们仅仅对如何智能地思考感兴趣，只要它能带来更好的表现。思想的作用是影响动作。

人工智能的核心工程目标是设计和合成有用且智能的产品。我们通常都是希望构建有智能行为的 Agent。这些具有智能行为的 Agent，可以使其许多应用大放光彩。

人工智能与自然智能

人工智能在本领域是一个固定的名称，但是“人工智能”这个术语同时也是一个很模糊的概念，因为人工智能可以看做自然智能的对立面。

对于任何现象，可以分成真与假两个对立面，假就是不真实；也可以分成自然与人工两个方面，自然意味着发生在自然界中的现象，人工则意味着由人类制造的那部分现象。

【例 1-1】 海啸是指由地震或山崩引起的海洋中的巨大波浪。自然海啸时有发生，人工海啸则是人为制造的，比如在海洋中投入一颗炸弹爆炸，这引发的仍然是一场真实的海啸。假的海啸可以是人工的，比如计算机软件绘制而成的海啸图形；也可以是自然的，比如显示有海啸发生的海市蜃楼现象。

对于智能是否不一样仍然存在争议：假的智能是不存在的。如果一个 Agent 可以智能地进行某种行为，我们说它是智能的。定义智能仅仅是外部的行为，正在智能地进行某个动作才是智能。因此，只有当智能被实现时，人工智能才是真的智能。

图灵受从外部行为定义智能这个概念的启发，设计了一个关于智能的测试 Turing [1950]，这就是著名的图灵测试。图灵测试是一种模仿博弈，测试者可以通过文本交互界面面向被测试者提出任何问题，如果测试者不能分辨出被测试者是不是人，这个被测试者就是智能的。图 1-1 显示了图灵测试中的一段对话。如果不能对任意话题表现出智能，这个 Agent 就不是真的智能。

测试者问：在你的十四行诗的第一行有一句话是“我能把你比作夏天吗？”，其中“夏天”换成“春天”是否可以或更好？

被测试者答：那样不合韵律。

测试者问：那换成“冬天”呢？它合韵律。

被测试者答：可以，但是没有人想被比作冬天。

测试者问：你大概说过 Pickwick 先生使你想起圣诞节吧？

被测试者答：从某种意义上讲是这样。

测试者问：但圣诞节是冬季的一天呀，我认为 Pickwick 先生不会介意比作冬天。

被测试者答：你也许是在开玩笑吧。我们所说冬季的一天是指冬季典型的一天，而不是像圣诞节那样特殊的一天。

图 1-1 图灵测试中的一段对话(来自 Turing[1950])

人们对图灵测试有很多争议。可惜的是，图灵测试虽然能为如何辨别智能提供测试，却不能为达到某种智能提供方法，因此图灵测试似乎并不是一种有用的研究思路。

最明显的自然智能 Agent 是人类。有人说蠕虫、昆虫或细菌是智能的，更多人说狗、鲸或猴子是智能的(见习题 1.1)。还有一类 Agent 比人类更为智能，那就是组织。蚁群就是组织的一个典型例子。单只蚂蚁可能不很智能，但蚁群却比任何一只蚂蚁更为智能。蚁群能发现食物并有效地利用，而且能很好地适应变化的环境。类似地，公司可以通过汇总所有的必要技能，对产品进行开发、制造和分配，这比个人单打独斗要有效得多。现代计

算机,从底层的硬件到高层的软件,其复杂程度常人无法想象,但每天却被人类所操作。人类社会可以看做迄今为止最为智能的 Agent 了。

人类智能从何而来是值得探索的。主要源头有以下三个方面:

生物:人类已进化成为可以在各种环境下生存的适应性动物。

文化:文化不仅能够提供语言,还能提供有用的工具、思维以及一代代从父母和老师传到孩子积累下来的智慧。

终生学习:人类从生活中学习和积累知识与技能。

这些源头以复杂的方式进行交互。生物进化论提供了生命不同阶段的不同学习而实现的成长。人类与文化是共同进化的,所以人类在刚出生时是没有能力的,大概就是因为我们有照顾婴儿的文化。文化和学习密切交互在一起。终身学习中的最主要部分是从父母和老师那里学到知识。语言,作为文化的一部分,使世界有所区别,同时也是学习的重要部分。

1.2 人工智能简史

纵观人类历史,人们曾一直用技术为人类自身建立模型,例如古代的中国、古埃及和古希腊都可以证明,每一种新技术都不断地用来建造 Agent 或其他意识模型。发条装置、水力学、电话转换系统、全息图、模拟计算机以及数字计算机等都被提出作为智能装置和意识模型的技术原理。

6

大约 400 年前,人们开始解释思维与推理的本质。Hobbes(1588—1679)被 Haugeland [1985, P. 85]称为人工智能的始祖,认为思考是符号推理,例如大声说话或用纸笔计算出某个答案。Descartes(1596—1650)、Pascal(1623—1662)、Spinoza(1632—1677)、Leibniz(1646—1716)以及精神哲学领域的其他先驱者对符号推理这一理念进行了更深层次的研究。

随着计算机的发展,符号运算这个概念变得更为具体。第一台通用计算机是一个分析机,它是由 Babbage(1792—1871)设计的,但直到 1991 年才真正制造出来,现在位于伦敦科学博物馆内。在 20 世纪早期,进行了很多关于认知计算的研究,提出了几种计算模型,例如 Turing(1912—1954)提出的图灵机——一种在无限长的磁带上书写符号的理论机器,Church(1903—1995)提出的 λ 演算——一种重写公式的数学形式体系,这种理论表明不同的数学形式其实是等价的,因为任意一个可计算的函数可以由其他函数计算得到。由此,我们得到 Church-Turing 理论:

任何有效可计算的函数都可以通过图灵机来运行(同样适用于 λ 演算或其他等价的形式体系)。

这里“有效可计算的”是指那些定义明确的操作;图灵机时代的“计算机”是执行明确定义步骤的人,而我们现在所熟悉的计算机那时并不存在。此理论表明所有的计算都可以通过图灵机或其他与之等价的计算机来执行。我们虽然无法证明 Church-Turing 理论,但它确实是一种经得住时间检验的假设。因为到现在都没有人可以制作出一种机器,来进行图灵机无法进行的计算,而且也没有证据说明存在某种函数,人类可以计算而图灵机却不能进行计算。Agent 的动作是一个关于其能力、历史、目标及偏好的函数。这也证明计算不仅仅是一个智能的暗喻;推理是可以通过计算机来执行的一系列计算。

计算机出现之后,最早期的某些计算机应用是人工智能程序。例如,Samuel[1959]在 1952 年开发的国际跳棋程序,在 20 世纪 50 年代末期,人们可以通过这样的程序来学习下

国际跳棋。Newell 和 Simon[1956]开发的程序“逻辑理论家”可以证明命题逻辑中的定理。

除了研究高级的符号推理,还有一些工作研究低级学习,这些研究从神经元如何工作受到启发。McCulloch 和 Pitts[1943]说明了一个简单的阈值“形式神经元”如何作为图灵完全机器的基础(图灵完全指在可计算性理论中,编程语言或任意其他的逻辑系统具有等价于通用图灵机的计算能力。换言之,此系统可与通用图灵机互相模拟。图灵完全性通常指具有无限存储能力的通用物理机器或编程语言。——译者注)。最早提出神经网络的人是 Minsky[1952],早期关于神经网络标志性的研究工作是 Rosenblatt[1958]的感知机。而 1968 年之后,神经网络的研究工作面临衰退,因为 Minsky 和 Papert[1988]认为这种学术表达不胜任智能行为。

7

早期的应用程序主要关注该领域的基础方面:学习和搜索。怎么描述解决问题需要的知识是要解决的主要难题之一,这个问题变得越来越明显。在学习之前,Agent 需要有适合已知知识的目标语言。有很多表达方法,从基于特征的表达到 McCarthy 和 Hayes[1969]的复杂的逻辑表达,还有很多介于两者之间的表达方式,如 Minsky[1975]提出的框架表达。

20 世纪六七十年代,在一些限定领域中建立的自然语言理解系统取得了成功。例如, Daniel Bobrow[1967]的 STUDENT 应用程序,可以用来解决用自然语言表示的高中代数问题。Winograd[1972]的 SHRDLU 系统,可以利用受限的自然语言讨论和解决模拟积木世界的任务。CHAT-80[Warren 和 Pereira, 1982]可以回答一些用自然语言表达的地理问题。图 1-2 展示了 CHAT-80 可以回答的一些问题,它的回答是建立在国家、河流等事实数据库的基础上。这些系统仅仅是在有限的领域内利用受限的词汇和语句结构来进行推理的。

8

阿富汗与中国相邻吗?
上沃尔特的首都是哪里?
哪国的首都是伦敦?
最大的非洲国家是哪个?
最小的美洲国家有多大?
与非洲国家及亚洲国家相毗邻的海洋是哪个?
与波罗的海相邻国家的首都有哪些?
多瑙河流经多少个国家?
赤道以南并且不属于澳洲^①的那些国家的总面积是多少?
每个大洲上国家的平均面积是多少?
每个大洲都多于一个国家吗?
流入黑海的河流发源于哪些国家?
不存在这样国家的洲有哪些?即人口超过 100 万的城市多于两个的国家。
哪个地中海沿岸的国家与人口超过印度的国家相邻?
哪些国家与大西洋相邻,并且人口超过 1000 万?

图 1-2 CHAT-80 可以回答的一些问题

在 20 世纪七八十年代,关于专家系统有大量的研究工作,目的是获取某个领域内专家的知识,以便于用计算机来执行专家任务。例如,从 1965 年到 1983 年在有机化学领域开发的 DENDRAL 专家系统[Buchanan 和 Feigenbaum, 1978],可以为新型有机化合物提出理论结构。从 1972 到 1980 年开发的 MYCIN 专家系统[Buchanan 和 Shortliffe, 1984],可以进行血液传染性疾病的诊断,制定抗生素治疗方案并解释它的形成原因。同期,人工智能推理得到了

① 此处“澳洲”指“澳大拉西亚”,一般包括澳大利亚、新西兰及附近南太平洋诸岛。——译者注

迅速发展,例如 Prolog 语言[Colmerauer 和 Roussel, 1996; Kowalski, 1988]。

在 20 世纪 90 年代和 21 世纪初,人工智能的子学科迅速发展,例如认知学、概率推理、决策理论推理、规划、嵌入式系统、机器学习以及其他很多学科。本领域的基础部分获得了进一步发展,这也正是本书的基础组成部分。

与其他学科的关系

人工智能是一门非常年轻的学科,而像哲学、神经生物学、进化生物学、心理学、经济学、政治学、社会学、人类学、控制工程学等学科研究智能已经很久了。

人工智能学科可以描述为“人造心理学”、“实验性哲学”或“计算认识论”,其中认识论主要研究知识。人工智能可以看做研究知识与智能本质这个老问题的一种方式,但采用了比以前更有效的实验工具。不同于哲学、心理学、经济学和社会学等只是传统地观察智能系统的外部行为,人工智能研究者们则对智能行为的可执行模型进行实验验证。更重要的是,这些模型还可以检查和重新设计,并以一种完全和严格的方式进行实验。现代计算机可以构造模型,而哲学家只能建立理论。人工智能研究者可以对这些模型进行实验,而不仅仅是讨论它的抽象性质。人工智能理论可以作为其实现的基础。此外,我们常常会对简单的 Agent 表现出复杂的行为感到非常惊讶,而这些是我们没有实现这些 Agent 时所想不到的。

对比过去几个世纪里飞行器的发展与过去几十年里计算机的发展,是非常有建设性意义的。我们通过几个方面来理解飞行,一种是仔细剖析已知的飞行动物,并将它们的共同结构特征假设为任意飞行 Agent 的必要基本特性。通过这种方法,检查鸟类、蝙蝠及昆虫的共同特征发现,飞行需要翅膀的拍动,而翅膀上面需有羽毛或薄膜。此外,这个假设可以通过实验来验证,就像伊卡洛斯做的那样,将翅膀捆绑在手臂上,拍动,然后跳到空中。另一个可行方法就是试图理解飞行的原理,不仅仅局限于自然界中的飞行。它涉及嵌入假设原理的人工产品结构,除了能飞之外,其他地方与飞行动物根本没有类似之处。通过第二种方法不仅制造出了飞机这样有用的工具,还产生了航空动力学这门能更好地理解飞行原理的学科。

人工智能采用和航空动力学类似的方法。人工智能研究者们对验证智能本质的一般假设很感兴趣,而制造智能机器并不一定要模仿人类或组织。通过考虑“飞机真的能飞吗?”这样类似的问题,给我们提供了解决“计算机真的能思考吗?”问题的一个思路。

人工智能与计算机科学紧密相连。尽管人工智能领域的很多研究者不是计算机科学家,但即使不是大部分,很多的人工智能研究都需依靠计算机科学才能完成。由于计算的研究是人工智能的中心,因此这是合理的。理解算法、数据结构及组合复杂性对于构建智能机是非常重要的。人工智能也为计算机科学带来了许多衍生利益,从分时系统到计算机代数系统等。

最后,人工智能可以看做认知科学的一部分。认知科学连接着多门研究认知与推理的学科,如心理学、语言学、人类学和神经学等。人工智能区别于认知科学的地方是它能提供工具构造智能,而不仅仅是研究 Agent 的外部行为或分析智能系统的内部运行原理。

1.3 环境中的 Agent

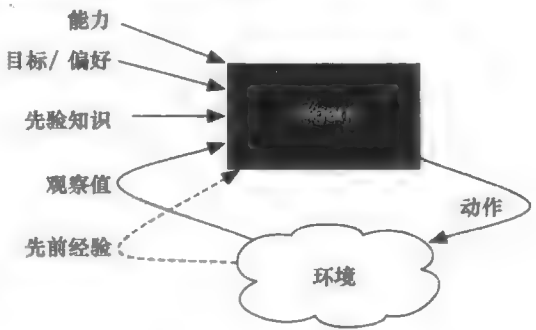
人工智能研究具有实用性的推理:为了完成某项任务或达到某一目的而进行的推理。Agent 由感知、推理和行为组成。Agent 在环境中进行某种行为,这里的环境也可以包括

其他的 Agent。一个 Agent 与其周围的环境统称为一个世界。

例如，被称做机器人的 Agent，是由带有物理传感器的计算引擎和制动器组成的，其行为环境是物理环境；再者，能够提供建议的计算机专家系统，能够感知信息以及执行任务；Agent 也可以只是存在于纯粹的计算环境中的程序，如软件 Agent。

图 1-3 展示了 Agent 的输入输出。任何时候，Agent 所做的都依赖于以下几个方面：

- 1) Agent 及其环境的先验知识。
- 2) 与环境的交互历史，其中包括：
 - 当前环境的观察值。
 - 先前经验和观察值，或从学习中获得到的其他数据。
- 3) 试图达到的目标或整个世界状态的偏好。



10

图 1-3 Agent 与环境的交互

4) 它能够执行的最原始动作，即能力。

两个具有相同先验知识、历史、能力和目标的 Agent 才会产生同样的行为，改变其中任何一个条件都将导致不同的行为结果。

每个 Agent 都有一些内部状态，能为它的环境及其自身进行编码。Agent 可能会有多个要达到的目标，在环境中为达到这些目标而存在多种行为方式，以及通过推理、感知和学习来改变信念的多种方法。纵观所有 Agent，从恒温控制器到一组移动机器人，到由人类提供感知和行为的诊断建议系统，再到社会本身，其复杂度各不相同。

1.4 知识表示

一般情况下，要解决的问题或要完成的任务，包括解的构成，是通过非形式化的方式给出的，例如“他们到达时，请立即递交包裹”或“请修理家中存在故障的电力系统”。

计算机解决问题的一般框架在图 1-4 中给出。为了解决问题，系统设计者必须：

- 具体化任务，并制定解决方案。
- 用特定的语言表达问题，以便计算机进行推理。
- 用计算机计算出相应结果进行输出，可以给用户呈现一个答案或是在环境中需要执行的一系列行为。
- 解释作为问题的解决方案的输出结果。

11

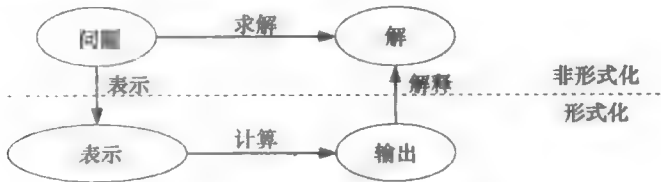


图 1-4 问题解决中的表示规则

知识是可以用来解决本领域问题的领域信息。解决多种问题需要多种知识，而且知识必须能够在计算机中进行表达。为了解决问题，设计应用程序时，我们需要为如何表达知识来进行定义。表示方案是 Agent 中使用的知识形式，而一部分知识的表示则是知识的内部表达。表示方案指定了知识的形式。知识库则是 Agent 中存储的所有知识的表达。

一个好的表示方案是很多竞争性目标间的折中。表示应该：

- 足够丰富，能够表达解决问题所需的所有知识。
- 尽可能地接近问题，并且简洁、自然、可维护。我们应该很容易地看出该表达与要表达的领域之间的关系，以便很容易地确定该表达是否正确。问题的微小变化，也应导致问题表达的微小改变。
- 能够经得起有效计算的检验，通常指能够有效地描述问题特征，有利于获得计算增益，并且能够在准确度和计算时间之间进行权衡。
- 能够从人类、数据或先前经验中获得。

已经设计出了不同的表示方案，很多方案开始时有某些目标，后来扩展到包含其他目标。例如，有些表示方案设计出来用于学习，而后被扩充用来解决更多的问题，并具有了推理能力。有些表示方案设计出来用于表达意识，而后又加上了推理及学习。有些表示方案设计出来用于推理，而后逐渐变得更为自然、易懂。

12

当给了一个问题或任务时，以下一些问题必须要考虑到。

- 问题的解决方案是什么？怎么样的解决方案才算好的？
- 问题如何进行表达？解决问题时需要世界中哪些特性？需要哪些具体的世界知识？Agent 如何从专家那里或以前的经验中获得知识？这些知识如何被调试、维护和改进？
- Agent 如何计算输出结果，以此作为问题的解决方案？能不能把最坏情况下或平均情况下的临界时间减到最小？理解答案是怎样推导出来的对于一个人来说很重要吗？

这些问题会在下一部分讨论，本书的表示方案部分也会涉及这些问题。

1.4.1 定义解

给定问题的一个非形式化描述，在考虑使用计算机之前，知识库设计者应该确定问题解的构成。不仅是人工智能领域，任何涉及软件设计的地方都会存在这个问题。很多软件工程都涉及问题描述的提炼。

通常，并不是所有问题都能很好地说明，不仅是有些地方没有进行说明，而且没有说明的地方也不能被随意填充。举例来说，当你让交易 Agent 搜索出那些有健康隐患的度假村时，你并不希望它返回所有的度假村信息，尽管符合你要求的度假村也包含在里面。然而，如果 Agent 根本就没有度假村的完全知识，那么返回所有的信息可能是保证符合要求的唯一方法。同样，当要求将垃圾传送到垃圾桶时，传送机器人却把所有的物品都传送到了垃圾桶里，尽管保证了所有垃圾都能被带走，但这样一个传送机器人你不会想要。人工智能的很多成果来自于常识推理；我们希望计算机能够对未阐明的假设做出常识性的推论。

给定一个定义明确的问题，下一步就是判断如果答案不正确或不完全，对问题的解决有没有影响。例如，如果说明书要求所有的实例，那么缺少几个实例可以吗？如果增加另外的实例可以吗？通常人们想要的是符合某些标准的最优解决方案，而不仅仅是任意方案。一般分为以下 4 类解决方案：

- **最优解。**问题的最优解是指在所有的可能解中最符合求解质量度量的最好的解。这个度量用序数来描述。但是在一些情况下，例如综合多种标准或不确定推理时，我们需要一种要考虑相对大小的基数度量。序数度量的一个例子是让机器人带出

13

尽可能多的垃圾，越多越好。作为基数度量的例子，你可以让传送机器人在最小化传送距离的同时，将尽可能多的垃圾带到垃圾桶，明确规定它们之间要进行权衡。为了节省时间，丢掉一些垃圾可能会更好。一般基数度量的期望值，即**效用函数**主要应用于决策论中。

- **可满足解**。Agent 经常并不需要最优解，只需一些满足条件的解就可以。**可满足解**是指那些满足充分条件的解。例如，人们可以命令机器人带出所有的垃圾，也可以命令它只带出三件垃圾。
- **近似最优解**。好的基数度量的一个优点是它允许使用近似值。**近似最优解**是指其质量度量接近理论上存在的最优解。一般 Agent 不需要问题的最优解，只要那些解接近最优就可以。例如，机器人为了送出垃圾可能并不需要达到最优的传输距离，可能只需要在最优距离 10% 范围以内就可以。

对于一些问题，获得近似最优解比获得最优解更容易，但对于另外一些问题，获得近似最优解和获得最优解一样难。一些近似算法能够得到在最优解的一定范围内的解，而有的则根本找不到。

- **可能解**。**可能解**是指一个解，尽管它可能不是问题的真正解，但也是一个有希望的解。这是一种用精确的方式来对可满足解进行近似的方法。例如，在传送机器人可能会丢垃圾或者无法捡起来的状况下，可以设置成机器人有 80% 的可能性捡起三件垃圾。我们一般期望去区分错肯定误差率(计算机给出的不正确答案的比率)和错否定误差率(计算机没有给出的正确答案的比率)。有些应用程序可能会容忍这些误差中的一个远超过另一个。

这里的分类并不是互相排斥的。有一种称为可能近似正确(PAC)的学习形式，认为可能学习到一种近似正确的概念。

14

1.4.2 表示

如果你对解的本质有要求，你就需要表示问题，以便于计算机来解决它。

计算机和人类思维都属于**物理符号系统**。**符号**是一种可以被操作的有意义的图案。例如，字、句、手势、文章里的标记与二进制序列都是符号。**符号系统**对符号进行创造、复制、修改和撤销。本质上来说，符号是符号系统中被操作的基本单元。

这里用到物理这个术语，是因为物理符号系统中的符号是部分现实世界中的物理对象，尽管它们可能存在于计算机和大脑内部。它们需要物理地影响行为或运动控制。

很多人工智能技术依赖于 Newell 和 Simon[1976]提出的**物理符号系统假设**：

物理符号系统具有必要且充分的机制以进行一般的智能活动。

这是一个强有力的假设，它的意思是任意 Agent 必定是一个物理符号系统，而且物理符号系统包括智能行为所需的一切，没有不可思议的量子论现象。世界中的物理符号系统也需要一个用来进行感知和行为的主体。物理符号系统假设是经验性的假设，像其他科学假设一样，它也需要证据来论证，或者有其他可替代的假设存在。实际上，它也可能是错误的。

Agent 可以看做通过操作符号来产生动作。其中许多符号用来表示世界中的事物；一些符号则用来表示有用的概念，这些概念可能具有扩展含义，也可能不具有扩展含义；另外还有一些符号表示 Agent 的内部状态。

Agent 可以用物理符号系统对世界进行建模。**世界模型**是世界中真实存在的详情和世界的动态行为的表达。世界并没有什么特定的标准。所有的模型都是**抽象**出来的，它们表

达的仅仅是世界的一部分，遗留了很多其他细节。Agent 可以有一个非常简单的世界模型，也可以有一个非常详细的世界模型。**抽象的层次**是指按抽象程度排列的一个偏序。低层抽象比高层抽象拥有更多的细节。Agent 可以拥有多重的，甚至是对立的世界模型。模型，不是看它是否正确，而是看它是否有用。

15

【例 1-2】 传送机器人可以对环境进行高层抽象建模，包括房间、走廊、门和障碍物，忽略距离、机器人的大小、需要的操纵角度、车轮的移动、包裹的重量、障碍物的详细情况、加拿大的政治形势以及其他事物。机器人也可以通过将一些细节加入模型中对环境进行低层次抽象建模。这些细节中的一些与机器人完成任务无关，而一些则是机器人成功的关键。例如，在某些情况下，为了使机器人不在一些特定的拐角处被卡住，机器人的大小和操纵角度是关键。而在其他情况下，如果机器人接近走廊的中心位置，那么就不需要对它们的宽度或操纵角度进行建模。

选择一个合适的抽象层次比较难，因为

- 对于人类来说，高层次的描述更容易说明和理解。
- 低层次的描述更准确，更有预测性。通常高层次抽象忽略的细节可能正是解决问题的关键之处。
- 层次越低，越难进行推理。因为低层次模型的解涉及更多的步骤和更多可供选择的行动路线。
- 你可能根本不知道低层次建模所需要的信息。例如，传送机器人可能不知道自己会遇到什么样的障碍物或地板有多滑，并在遇到这些情况时应该做些什么。

对环境进行多层次抽象建模可能会更好些。这个问题会在 2.3 节进行讨论。

我们可以对生物系统和计算机进行多层次抽象描述。逐层降低的层次是神经层次、生物化学层次(什么样的化学药品和电势可以传送)、化学层次(会进行什么样的化学反应)、物理层次(原子力和量子论现象)。在神经层次之上需要什么层次来计算智能仍是一个开放性难题。注意，这些层次描述在科学本来的分层结构中得到共鸣，科学家被划分为物理学家、化学家、生物学家、心理学家、人类学家等。尽管我们不知道哪个层次的描述更重要，但我们猜想建立一个 Agent 不必模仿人类的每一层，可以在现代计算机的基础上进行高层次建模。这个猜想是人工智能研究的一部分。

对生物实体和计算性实体来说，以下两个层次看起来是共有的：

16

- **知识层**是将 Agent 所知道的、所相信的以及它的目标抽象出来。知识层考虑的是 Agent 知道什么，而不是它如何进行推理。例如，传送 Agent 的行为可以用它是否知道包裹有没有到达和是否知道特定的人在哪里来描述。人类和自动 Agent 都可以在知识层次上进行描述。在这个层次上，不需要说明如何计算解和 Agent 可能会用到哪些策略。
- **符号层**是对 Agent 的推理进行描述。为了实现知识层，Agent 需要通过操作符号来产生答案。我们设计了很多认知科学方面的实验来判定推理时会产生什么样的符号操作。注意，知识层说明的是对于外部世界，Agent 相信什么，它的目标是什么；而符号层是对 Agent 内部关于外部世界推理的描述。

1.4.3 推理与行为

符号的操作产生行为被称为推理。

人工智能表达不同于用传统语言编写的计算机程序的一个重要方面是，人工智能表达

的主要是需要计算什么，而不是怎样进行计算。我们可能明确指出 Agent 应该找出病人最可能得的疾病，或指出机器人应该拿杯咖啡，而不是给出做这些事情的具体指令。很多人工智能推理涉及通过可能空间的搜索来决定怎样完成任务。

在决定 Agent 要做什么时，有三个方面的计算必须区别开来：1) 要设计 Agent 时的计算；2) 在观察世界、需要行动之前，Agent 能做的计算；3) 当 Agent 执行某些动作时所做的计算。

- **设计时刻推理**是在设计 Agent 时进行的推理。它由 Agent 的设计者而不是 Agent 本身来完成。
- **离线计算**是在动作之前由 Agent 完成的计算，它包括汇编和学习。Agent 离线获取背景知识和数据，并将其汇编成可用的形式，称之为**知识库**。背景知识可以在设计时或离线状态时给出。
- **在线计算**是在观察环境及在环境中进行某些行为时由 Agent 完成的计算。在线状态获取的信息称为**观察**。Agent 必须使用知识库和观察来决定要做什么。

区别设计者思维中的知识和 Agent 思维中的知识是重要的。考虑以下极端情况：

- 一种极端情况是，高度专业化的 Agent 在为其设计的环境中能够运行良好，但离开这个合适位置却变得无能为力。设计者可能为构造 Agent 做了大量工作，而 Agent 本身不用做很多就可以运行良好。举个自动调温器的例子，设计一个自动调温器是困难的，因为它必须在正确的温度下准确打开或关闭，但是自动调温器本身并不需要做很多计算。再比如，绘图机器人每天在汽车厂绘制相同零件的例子。为了完美地完成绘制任务，可能需要很多设计时间或离线计算，但绘图机器人几乎不需要在线计算就能完成绘图任务；它能感知到有一个零件在特定的位置，然后进行其预先定义好的动作。这些专业 Agent 不适用于不同的环境或变化的目标。如果一个不同的零件放在特定的位置，绘图机器人不会注意到，即使注意到了也不知道要做些什么，必须通过重新设计或重新编程来绘制不同的零件，或者将其改变成为磨光机或跟踪洗涤剂。
- 另一种极端情况是非常柔性的 Agent，可以在任意环境下生存，可以在运行时接受新任务。简单的生物 Agent，如昆虫，能适应复杂变化的环境，但是它们不能执行任意任务。设计一个可以适应复杂环境和变化目标的 Agent 是一个主要挑战。Agent 能够比设计者知道更多的环境细节。就算是在生物学方面也没有制造出这样的 Agent。人类可能是唯一现存的实例，但即使是人类也需要时间来适应新环境。

虽然柔性 Agent 是我们的终极梦想，但研究者们也不得不通过更多的平凡目标来达到这个目标。研究者们已经制造了很多适应于特定环境下的特定 Agent，而不是制造那种能够适应任何环境完成任何任务的通用型 Agent。设计者能够对特定环境结构进行开发，而 Agent 不必对其他的可能性进行推理。

构造 Agent 的两大一般策略如下：

- 第一个策略是简化环境并为这些环境建立复杂的推理系统。例如，工厂机器人可以在工厂这个工程环境里完成各种复杂的任务，但它们在自然环境中可能是没用的。很多问题的复杂性可以通过简化环境来降低。建立实用性的系统也是非常重要的，因为许多环境可以为 Agent 设计得更简单些。
- 第二个策略是在自然环境中建立简单的 Agent。这种策略是受到昆虫的启发，昆

虫虽然只有有限的推理能力却能在复杂环境中生存。随着它们的任务越来越复杂, Agent 可以被设计为拥有更多的推理能力。

简化环境的一个优点是我们能用它证明 Agent 的性质或在特定状态对 Agent 进行优化。证明 Agent 的性质或优化主要需要一个 Agent 及其环境的模型。Agent 可能会做一点或很多推理, 但 Agent 的观察者或设计者能够对 Agent 及其环境做出推理。例如, 设计者能够证明 Agent 是否能够达到目标, 是否能够避免陷入不利的状况中(安全目标), 会不会在某个地方卡住(活跃性), 会不会最终开始它应该做的事情(公平)。当然, 证明实际上等价于其模型。

为复杂环境构造 Agent 的优点是人类的生活环境和我们想要 Agent 活动的环境是多种多样的。

幸运的是, 这些方面的研究都一直在进行。在第一种情况下, 研究者们从简单的环境入手, 逐步使环境变得更加复杂。在第二种情况下, 研究者们增加 Agent 行为的复杂性。

1.5 复杂性维度

从自动调温器到在竞争性环境中有多重目标的企业, Agent 在环境中行为的复杂性各不相同。Agent 的设计存在多个维度的复杂性。这些维度可以分开来考虑, 但建造智能 Agent 时必须组合起来。这些维度定义了人工智能的一个设计空间, 空间里的不同点可以通过改变维度值来得到。

这里我们介绍 9 个维度: 模块性、表示方案、规划期、感知不确定性、效用不确定性、偏好、Agent 数量、学习和计算限制。这些维度对智能系统的设计空间做了粗糙的划分。有时为了建立智能系统必须做出很多其他的选择。

1.5.1 模块性

第一维是模块性的层次。

模块性是系统可以分解成能被独立理解的交互模块的程度。

模块性对降低复杂性是重要的。它在脑部结构中很明显, 是计算机科学的基础, 也是任何大型组织的重要部分。

模块性主要通过层次结构分解体现。例如, 人的视觉皮层与眼睛组成了一个模块, 可以采光, 还可以达到更高层的目标, 输出一些简化的情景描述。如果模块再分成更小的模块, 并以此类推, 还可以分成更小的模块, 直至分成最原始的操作, 我们说模块性是分层的。这种分层组织正是生物学家研究的一部分。大型组织都有一个分层结构, 这样高层决策者才不会被细节压倒, 也不必过问所有的组织细节。计算机科学中的抽象编程和面向对象编程, 就是利用模块性与抽象性, 使系统更为简化。

在模块性维度中, Agent 的结构是下列一种:

- **扁平的:** 系统中没有组织结构;
- **模块化的:** 系统可被分解成独立的、可理解的交互模块;
- **分层的:** 系统是模块化的, 模块本身分解成了交互模块, 它们中的每一个又都是一个分层系统, 由此循环下去, 直到分解成最简单的组件。

在扁平结构或模块化的结构中, Agent 主要进行单层抽象上的推理, 而在分层结构中, Agent 主要是在多层抽象上进行推理。低层结构主要涉及低层抽象的推理。

【例 1-3】 一个 Agent, 比如你自己, 从家中到海外的度假目的地去旅行, 必须从家里到机场, 然后飞到目的地附近的机场, 再从飞机场到达目的地。它还可以将实际的移动分解成一系列的腿部运动或车轮滚动。在扁平结构表达中, Agent 会选择某一层抽象表达并在该层进行推理。模块化表达将任务分解成很多可以独立解决的子任务(例如, 订票, 到达出发地机场, 到达目的地机场, 到达度假的地点)。在分层结构表达中, Agent 将用分层的方式解决这些子任务, 直至将问题分解成一些简单的问题, 例如传送 http 请求或采取一种特定的步骤。

对于降低构建能在复杂环境中动作的 Agent 的复杂性, 层次分解是重要的。然而, 为了考察其他维度, 我们首先忽略分层结构, 而假设它是一个扁平结构。忽略层次分解这种做法通常适用于中小型问题求解, 它适用于低等动物、小型组织或中小型的计算机程序。当问题或系统较为复杂时, 就需要一些分层结构了。

如何建立具有分层结构的 Agent 将在 2.3 节进行讨论。

1.5.2 表示方案

表示方案维度主要讨论如何描述世界。状态是指世界中影响 Agent 行为的各种方法。我们可以将世界的状态分解成 Agent 的内部状态(它的信念状态)和环境状态两个部分。

在最简单层次上, Agent 可以以一系列独立的确定状态来进行推理。

20

【例 1-4】 加热器的自动恒温器一般有两个信念状态: 关闭和加热。环境有三个状态: 冷、舒适和热。信念状态与环境状态的不同组合, 可以组成相对应的六种状态。这些状态可能无法完全形容世界, 但它们足以来描述自动恒温器的所有行为了。如果环境处于冷状态, 自动恒温器应该调到或停留在加热状态; 如果环境处于热的状态, 自动恒温器应该调到或停留在关闭状态; 如果环境处于舒适状态, 自动恒温器停留在现在所在的状态下即可。Agent 在加热状态可以加热, 而在关闭状态则不能加热。

用状态特征或状态真假的命题进行推理, 要比列举一系列状态进行推理更容易些。状态可以用一系列的特征来描述, 其中每个特征在每个状态中都有一个值(见 4.1 节)。

【例 1-5】 用来看护房屋的 Agent 可能必须对灯泡是否被破坏进行推理。它可能有每个开关的位置、状态(是否工作良好, 是否短路, 是否被破坏)和每个灯泡是否能够正常工作的特征。特征 pos_s2 有这样的特点: 当开关 $s2$ 调上去时, pos_s2 的特征值为 up ; 当开关调下来时, pos_s2 特征值为 $down$ 。房间里照明设备的状态可以用这一系列的特征值来描述。

布尔型命题的值只有真、假两种情况。30 个命题经过编码可以有 $2^{30} = 1\,073\,741\,824$ 种状态。用这 30 个命题来阐述和推理可能比使用 100 多万种状态更容易些。另外, 用状态的紧凑表示法会更容易理解些, 因为它意味着 Agent 已经掌握了本领域内的一些规律。

【例 1-6】 考虑一个能够识别字母表里字母的 Agent, 假设这个 Agent 在观察一个 30×30 像素的二值图像, 这 900 个网格点中的每一个或开启或关闭(也就是, 没有使用任何色彩或灰度信息)。这个行为将决定图像中绘出的是字母 $\{a, \dots, z\}$ 中的哪一个。这个图像有 2^{900} 种不同的状态, 从图像状态到 $\{a, \dots, z\}$ 这 26 个字母的映射函数会有 $26^{2^{900}}$ 种。我们甚至无法用状态空间来表达这些函数。因此, 我们定义了图像的特征, 例如线段, 并用这一系列的特征定义了从图像到字母的映射函数。

当描述复杂世界时, 特征可以依赖于关系与个体。单个个体上的关系是一种属性, 个体之间的每一种可能关系上都存在特征。

21

【例 1-7】在例 1-5 中看护房屋的 Agent 可以将灯泡与开关看做个体，并有它们之间的位置关系 *position* 与有向连接关系 *connected_to*。用位置关系 *position(s₁, up)* 来表示 *position_s₁=up* 这个特征。当 Agent 遇到开关并且具有此开关的先验知识时，Agent 就能根据关系对这些开关进行推理。

【例 1-8】假设 Agent 是一个学生课程登记系统，用特征 *grade* 来描述一个学生所选的一门课程的成绩。对于每一组学生-课程对 (*student*, *course*)，存在特征 *passed*，特征 *passed* 依赖于特征 *grade*。用学生个体、课程个体、成绩个体以及 *grade* 和 *passed* 的关系来进行推理，可能会更容易些。通过定义特征 *passed* 是怎样依赖于特征 *grade* 的，Agent 便能将其应用于每个学生与每门课程。而且，在 Agent 知道任意个体及其所有特征之前，这些就能完成。

因此，用个体及他们之间的关系进行相关性描述，比对那些特征或命题进行处理，可能更为简便。例如，100 个个体和其二值关系可以用 $100^2 = 10\,000$ 个命题及 $2^{10\,000}$ 个状态来表示。通过一系列的关系与个体推理，Agent 可以只通过状态对全部类型的个体进行解释推理，而不用枚举特征或命题。Agent 有时不得不对无限个体集进行推理，如所有数的集合或所有语句的集合。对于无限个体集，Agent 无法用状态或特征进行推理，只能在关系层面上进行推理。

在表示方案维度中，Agent 可以通过以下几个方面进行推理：

- 状态；
- 特征；
- 从个体与关系角度进行的关系描述。

一些框架结构是通过状态来描述的，而有些则是通过特征完成的，另外有些则是建立在关系的基础上。

我们将在第 3 章介绍状态推理，在第 4 章介绍特征推理，关系推理将在第 12 章介绍。

1.5.3 规划期

下一个维度是用来说明 Agent 规划的向前时间的程度。例如，一只狗被叫过来，它会在未来能得到奖励而奔跑，而不仅仅是为了得到即时的奖励而行动。狗不会为了未来任意无限期长的目标而行动(例如几个月后)，而人类可以(例如为了得到明年的假期而努力工作)，这个说法看来是对的。

22

当 Agent 决定做什么时，能够观察到未来的远近，我们称之为规划期。也就是说，规划期是 Agent 认为它的动作结果所能影响的向前程度。从完备性上来说，包括 Agent 不能及时进行推理的非规划情况。我们把 Agent 做规划时所考虑的时间点称为阶段。

在规划期维度中，Agent 可以分为以下几种：

- 非规划 Agent，是指在决定做什么时，不考虑未来的影响，或者不涉及时间的 Agent。
- 有限期规划者，是指遵循固定有限时间步的 Agent。例如，医生治疗病人，但之前一般会花一些时间做一些检查，所以整个过程可以分为两个阶段来规划：检查阶段、治疗阶段。在退化状况下，Agent 可能只进行一个时间步，我们称它是贪婪的或目光短浅的。
- 不确定期规划者，是指能够向前探索几步但是不预先确定多少步的 Agent。例如，一个 Agent 必须到达一个位置，但是预先不确定到达那个位置需要多少步。
- 无限期规划者，是指一直在进行规划的 Agent。通常称之为过程。例如，腿式机

机器人上的稳定模块永远在运行；只有在达到稳定状态时才会停止，因为这种机器人永远在为了防止摔倒而维持稳定。

1.5.4 不确定性

Agent 可以假设没有不确定性，也可以把不确定性考虑进去。不确定性可以分为两部分：感知不确定性和效用不确定性。

1. 感知不确定性

在一些情况中，Agent 能够直接观察到世界的状态。例如，在一些棋类游戏中，或在工人工作的场地，Agent 能够精确地知道世界的状态。在许多其他情况中，Agent 能对世界的状态有一个带噪声的感知，它所能做的最好的就是在它所感知的状态集上建立概率分布。例如，给定一个病人的症状，医师实际上可能不知道病人患了什么病，而只有病人可能患有疾病的概率分布。

感知不确定性维度主要是用来说明 Agent 能否从观察中得到世界的状态。

- **完全可观察**，是指 Agent 能够从观察结果中得到世界的状态。
- **部分可观察**，是指 Agent 不能直接观察到世界的状态。出现这种情况可能是相同的观察结果导致很多可能的状态或是观察结果有噪声。

将世界假设为完全可观察的是一种简化假设，其目的是为了推理更容易进行。

2. 效用不确定性

在一些情况中，Agent 能够知道动作效果，也就是说，给定一个状态和动作，它能精确地预测出在那种状态下运行那种动作后的状态。例如，与文件系统进行交互的 Agent 能够在给定的文件系统的状态下预测出删除一个文件后的效果。在很多情况下，Agent 很难预测动作效果，最好也只能有一个效果的概率分布。例如，即使一个人知道狗的状态，他可能也不会知道对狗进行命令后的效果，但基于经验，他可能会知道这只狗可能会做些什么。即使是那些他以前没有见过的狗，狗的主人甚至也会知道在他发出命令时那些狗会做些什么。

效用不确定性从动力学方面可以分为：

- **确定性的**——动作所导致的状态由动作及之前的状态决定。
- **随机的**——对于结果状态，只能给出一个概率分布。

本维度只在世界完全可观察时成立。如果世界是部分可观察的，针对动作效果依赖于不可观察的特征的情况，随机系统可以建立一个确定性系统模型。它是一个单独的维度，因为我们建立的很多框架都是针对完全可观察的、随机动作的情况。

对确定性行为的规划将在第 8 章介绍，对随机动作及部分可观察域的规划将在第 9 章介绍。

1.5.5 偏好

Agent 会为自身获取更优的结果，做出某一动作优于另一动作选择的唯一原因是其偏好动作会导致更理想的结果。

一些 Agent 可能会有一个简单的目标，可能是要达到的状态或是要证明为真的命题，例如为主人拿一杯咖啡（在她有咖啡的状态时结束）。另外一些 Agent 则可能会有更为复杂的偏好。例如，医师一般会考虑痛苦、预期寿命、生命质量、金钱成本（对病人、医生和

社会)、在诉讼案例中为决定辩护的能力以及其他一些必要的东西。当这些条件发生冲突时,正如他们总是做的那样,医师必须对这些考虑做出折中处理。

偏好维度是看 Agent 是否有:

- **目标。**此目标可能是在某一最终状态下要达到的**完成目标**,或是在所有已访问过的状态中**必须被保持的目标**。例如,机器人的目标可能是拿到两杯咖啡和一只香蕉,并且在这期间不能制造混乱或伤害任何人。
- **复杂偏好。**复杂偏好涉及在不同时期权衡各种期望的结果。**序数(ordinal)偏好**就是只注重偏好的排序。**基数(cardinal)偏好**涉及有关值的大小。例如,比起黑咖啡来说,山姆更喜欢卡布奇诺,而比起茶来说,更喜欢黑咖啡,这是序数偏好。基数偏好给定等待时间与饮料类型间的权衡,各种味道之间的权衡,如果咖啡的味道特别好,Sam 就能在等待咖啡的过程中容忍时间的煎熬。

目标将在第 8 章介绍,复杂偏好将在第 9 章介绍。

1.5.6 Agent 数量

仅一个 Agent 在它所属的环境里进行推理就已经足够困难。然而,如果有多个 Agent 进行推理将更为困难。多 Agent 背景中的 Agent 应该具备对其他 Agent 进行策略性推理的能力,其他 Agent 可能会对它进行欺骗或操纵,也可能会与它进行合作。对于多 Agent,因为其他 Agent 可以采取确定性策略,因此最优的选择经常是随机动作。即使当 Agent 之间进行合作,具有共同目标时,协商与交流的问题也使得多 Agent 推理更具挑战性。然而,许多领域包含多个 Agent,而且忽略其他 Agent 策略的推理并不是最好的方式。

从单个 Agent 的角度来看,Agent 数量维度主要是考虑 Agent 是否进行:

- **单个 Agent 推理。**Agent 会假设其他的 Agent 为环境的一部分。如果没有其他的 Agent,或者其他 Agent 的动作不会因为这个 Agent 的动作而改变,那么这个假设是合理的。
- **多 Agent 推理。**Agent 会将其他 Agent 的推理考虑进来。当其他 Agent 的目标或偏好部分依赖于此 Agent 的行为,或 Agent 必须与其他 Agent 通信时,这种情况就会发生。

如果 Agent 同时进行动作,或环境只是部分可观察的,那么与其他 Agent 一起进行的推理将会更为困难。多 Agent 系统将在第 10 章介绍。

1.5.7 学习

在某些情况下,Agent 设计者可能为 Agent 建立了一个比较好的模型及环境。但通常情况下,Agent 设计者无法建立完美的模型,Agent 需要一些先验知识或其他的资源来帮助它进行决策。

学习维度由以下两方面决定:

- **已有的知识。**
- **学到的知识(从数据或先前经验中获取)。**

学习一般意味着要找到与数据相符的最好模型,有时候这就像调整固定参数集一样简单,但这也意味着要从一类表达中选择最优的表达。学习本身就是一个很宽泛的领域,但不是孤立于人工智能的其他领域。除了拟合数据外,还有其他很多问题,包括如何合成

背景知识, 需要搜集什么样的数据, 如何表达这些数据以及结果, 什么样的学习偏差是合适的, 怎样合理使用学习到的知识去影响 Agent 的行为。

学习将会在第 7、11 和 14 章介绍。

1.5.8 计算限制

有时候 Agent 可以足够迅速地决定它的最好的行为, 但通常会有很多计算资源限制, 阻碍实施这些最好的行为。也就是说, 由于 Agent 的内存限制, 尽管某个动作是最好的, 但是它可能不能够迅速地找到这个最好的动作。例如, 如果 Agent 必须现在动作, 在十分钟之前花费十分钟的时间来推理最好应该做什么, 就没有什么用处。通常情况下, Agent 必须权衡得到一个解所花费的时间和解的好的程度; 有时候迅速地找到一个合理解可能要优于花费更长时间来寻求一个更好的解, 因为在计算期间世界可能会改变。

计算限制维度由 Agent 是否具有以下性质来决定:

- 完全理性: Agent 可以推出最佳行动方案, 而不考虑有限的计算资源;
- 有限理性: 在给定的计算限制上, 决定它所能找到的最佳行为方案。

计算资源限制包括计算时间、内存和数值精度。其中数值精度的限制是由于计算机不能准确地表示实数而引起的。

任意时间算法是解的质量随时间的推移而提高的一种算法。实际上, 它可以在任意时间产生当前最佳解, 但如果给定更多的时间, 可能会产生更好的解决方案。通过允许 Agent 存储迄今为止发现的最佳解, 我们可以确保得到解决方案, 并且解决方案的质量不会下降。然而, 等待 Agent 动作需要花费一定的时间; 对于 Agent 来说, 在找到最佳解之前动作可能更好些。

【例 1-9】 图 1-5 说明了任意时间算法的计算时间是如何来影响解的质量的。Agent 实施一个动作时, 能做一些计算来决定要做什么。绝对的解的质量, 在时间零点执行的动作, 如顶端的短划线所示, 会随着 Agent 利用时间推理而提高。然而, 花时间去行动会有损失。在本图中, 如底部的虚线所示, 这个损失与 Agent 执行动作之前的时间成比例。这两个值相加就能得到折扣的质量, 依赖于时间的计算值, 这就是中间部分的实线。如图 1-5 所示, 一个 Agent 计算大约需要 2.5 个时间单元, 然后执行动作, 在该点折扣质量达到最大值。如果计算持续长于 4.3 个时间单元, 产生的折扣质量将会比仅仅输出算法的初始猜测而实际不进行计算时更糟糕。解的质量有一个跳跃性的提高是非常典型的; 当现有的最好解发生变化时, 解的质量会有一个跳跃性的变化。然而, 与等待相关的损失通常并不是一条简单的直线。

将有限理性考虑在内, Agent 必须决定是应该立即实施动作还是进行更多的思考。这是一项具有挑战性的难题, 因为 Agent 通常无法

确定, 当它仅花费多一点时间进行推理时, 到底会有多好。而且, 在考虑是否进行推理上

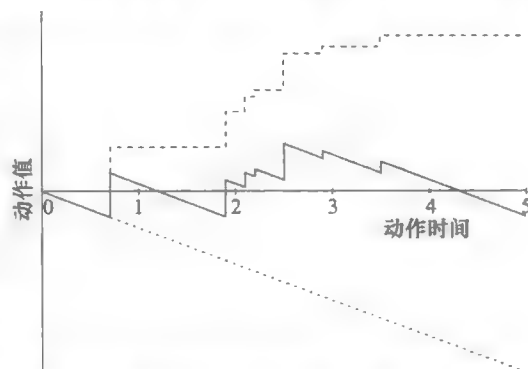


图 1-5 对任意时间算法来说, 解的质量是一个时间函数。Agent 必须选择一个动作。随着时间的推移, Agent 能够决定更好的动作。短划线表示的是, 如果它一开始就实施, Agent 迄今为止所能找到的最佳动作值。虚线表示 Agent 在等待行动时减少的行为值。实线表示 Agent 的净值, 是一个时间函数

花费的时间会减少实际推理的可用时间。无论如何，有限理性可以作为近似推理的基础。

1.5.9 多维交互

图 1-6 总结了一系列复杂性的维度。不幸的是，我们无法单独研究这些维度，因为它们以各种复杂的方式进行交互。在这里，我们给出一些交互实例。

维度	值
模块性	扁平的、模块化的、分层的
表示方案	状态、特征、关系
规划期	无规划、有限阶段、不确定阶段、无限阶段
感知不确定性	完全可观察、部分可观察
效用不确定性	确定性的、随机的
偏好	目标、复杂偏好
学习	已知的知识、学到的知识
Agent 数量	单个 Agent、多 Agent
计算限制	完全理性、有限理性

图 1-6 复杂性维度

表示维度与模块性维度进行交互。一些层次上的模块甚至简单到可以用一系列有限状态集进行推理，而其他的抽象层次可能要对个体及关系进行推理。例如传送机器人，维持平衡的那个模块可能仅有一小部分状态，而必须优先考虑将多个包裹传递给多个人的模块可能必须对多个个体(比如，人、包裹和房间等)及他们之间的关系进行推理。在更高层次上，对一天内执行的动作进行推理可能只需几个状态来概括这一天内的不同阶段(如，可能三个状态：忙时、可请求时和再充电时)。

规划期与模块性维度进行交互。例如，在较高层次时，跑过来并得到治疗后，小狗可能会得到即时的奖励。当决定将它的爪子放在哪里时，可能会需要很长时间来得到奖励，这时就不得不规划一个无限阶段。

感知不确定性可能对推理的复杂度有很大影响。对 Agent 来说，当它知道世界的状态时比不知道时更容易进行推理。尽管对状态的感知不确定性很好理解，但是对个体及关系的感知不确定性是现在研究的热点。

效用不确定性与模块性维度进行交互：在分层结构的某个层次上，某个动作可能是确定性的，但在另外的层次则可能是随机的。例如，你跟一个你正试图讨好的同伴飞到巴黎，在某个层面上来说，你可能知道你所在的位置(巴黎)，在较低层面上，你可能会完全迷失并不知道自己在哪儿。在负责维持平衡的更低层面上，你会知道你在哪儿：你正站在地上。在最高层面上：你可能根本不能确定是否给同伴留下了好印象。

偏好模型与不确定性交互，是因为 Agent 必须权衡满足有一定概率的主要目标，还是满足具有更高概率的次要目标。这个问题将在 9.1 节进行讨论。

模块性可以使用多 Agent。设计单个 Agent 的一种方式构造多个拥有共同目标的交互 Agent，这样能够使较高级别的 Agent 智能地执行动作。一些研究者，例如 Minsky [1986]，认为智能是非 Agent 社会的一个涌现特征。

学习可以通过特征来进行描述，决定哪个特征值能够最好地预测其他特征的值。然而，学习也可以通过个体及关系来进行。我们现在已在学习层次结构、在部分可观察领域

内学习和多 Agent 学习等方面做了很多工作。即使不考虑与其他多重维度的交互，这些研究方向本身也都具有很大的挑战性。

维度中模块性和有限理性能使推理更有效。虽然它们使形式化体系变得更为复杂，但却能够通过将系统分解成更小的组件并提供所需的近似值，使 Agent 能够在有限时间和有限的内存中及时地执行动作，来帮助构建更为复杂的系统。

1.6 原型应用

人工智能领域的应用广泛而且多样化，包括医疗诊断、工厂流程调度、险恶环境中的机器人、博弈、太空中的无人驾驶车辆、自然语言处理系统、指导系统等。这些应用并不是独立进行的，我们抽象出这些应用的本质特点，来研究智能推理及动作背后的原理。

本节概述了 4 个应用领域，其中的几个实例将贯穿整本书。尽管我们介绍的几个实例都很简单(如此他们才适用于本书)，但其应用领域代表了人工智能技术能够或正在应用的领域范围。

29

4 个应用领域如下所示：

- **自主传送机器人**会在某个建筑物旁徘徊，负责给里面的人传送包裹和咖啡。这种传送 Agent 能够找到路径、分配资源、接受人们的请求、决定优先顺序、在不伤害到人和它本身的情况下传递包裹。
- **诊断助手**帮助人们找出问题并提出修改或治疗意见来改正问题。例如电工助手，能够在给定电工问题特征的情况下就房屋中的错误提出建议，包括保险丝烧断、灯开关毁坏、灯泡烧坏等情况。再如医疗诊断助手，在特定医疗领域和病人的症状及病历等知识的基础上，可以发现潜在的疾病、有用的检查以及合适的治疗手段。对检测和修理它的人们以及那些要为它们的行为最终负责的人们，这种助手可以解释它的推理过程。
- **指导系统**与一个学生进行交互，展示某些领域的信息，并检查学生的知识或行为。这不仅仅是向学生展示信息。做到像一个好老师那样，根据每个学生的知识、学习偏好及误区，做到因材施教，展示不同的信息，这是一项更具有挑战性的工作。系统必须要理解所设计的科目以及学生的学习情况。
- 一个**交易 Agent**能够知道一个人想要什么，能买什么，并为他的利益服务。它应该知道他的需求和偏好，以及如何权衡竞争性目标。例如，如果一个家庭要去度假，旅游 Agent 需要预订宾馆、飞机票、出租车以及娱乐活动，所有这些都要相互适合。它应该决定顾客的权衡。如果最舒适的宾馆无法为这一家人在所有的假期内提供住宿，这时就要看他们是更喜欢在余下时间选择更好的宾馆，还是并不喜欢变换宾馆。它甚至可以为特价商品货比三家或一直等到更好的交易出现。

本书中所有实例都应用到上述 4 个领域，在以下几个小节我们将详细讨论每个应用领域。

1.6.1 自主传送机器人

我们来想象一个机器人，它具有多个轮子，能够捡起物体并可以放下。它具有感知能力，因此它能识别需要操控的目标物体，也能绕开障碍物。它可以以自然语言的形式接受命令，并遵循这个命令，能够在目标发生冲突时做出合理选择。这种机器人可以应用于办

公环境中, 来传送包裹、邮件、咖啡等物品, 或者嵌入轮椅中来帮助残疾人。它应该是有用且安全的。

在图 1-3 Agent 的黑盒特征中, 自主传送机器人有以下输入:

- 先验知识, 由 Agent 设计者提供, 关于 Agent 自身的能力, 比如 Agent 可能会遇到什么样的目标物并且区分这些物品, 请求的含义, 还可能有 Agent 的环境, 比如地图;
- 通过动作得到的先前经验, 例如, 它的动作效果, 世界中的常见对象是什么, 在同一天中的不同时段会有什么样的请求;
- 目标, 可以表示为应该传送什么, 什么时候传, 以及如何进行权衡, 例如什么时候它必须放弃一个目标去完成另外一个, 或者在执行动作的迅速性与安全性之间做出权衡;
- 观察值, 从相机、声波定位仪、触觉、声音、激光测距仪或键盘这样的输入设备中获取对环境的观察值。

机器人的输出是机械控制, 规定它的轮子如何转动, 它的四肢应该往哪儿移动, 它的钳子应该做什么。其他的输出还有语音和视频输出。

在复杂度的一系列维度中, 当机器人是扁平系统时是最简单的状况, 用一系列的状态来表达, 没有不确定性, 有可完成的目标, 没有其他 Agent, 有给定的知识, 而且具有完全理性。在这种情况下, 存在一个不确定阶段规划期, 决定要做什么的问题被简化成状态图中寻找一条路径的问题。这一部分将在第 3 章介绍。

对于推理任务来说, 每一个维度上都能增加概念复杂度:

- 层次分解, 通过将每个模块简单化而且能够被自身所理解, 从而使整个系统的复杂度增加。第 2 章将介绍这一部分。
- 用一系列的特征建模比用清晰状态建模使系统更容易理解。例如, 可能会有机器人的位置特征, 它所有的燃料数量特征, 它所运送的物品特征等。用一系列的状态进行推理, 其中每一个状态是对应于每个特征的值的分配, 失去了这些特征提供的结构信息。用一系列的特征表示来进行推理, 可能会开发计算增益。用一系列的特征进行规划将在第 8 章讨论。当涉及多个个体时(如多人或多个传送物), 用一系列的个体与关系进行推理可能会更容易些。用一系列的个体与关系进行规划将在 14.1 节讨论。
- 如果 Agent 仅仅是向前看几步, 它的规划期是有限的; 而如果它有一组固定的要达到的目标集, 那么它的规划期就是不确定的。随着不间断的需求和动作, 为了长期幸存, 它的规划期是无限的, 例如, 邮件到达时就传送, 电池电量低时就先进行充电。
- 可能存在目标如“将咖啡传送给克里斯, 并保证你自己一直有能量”。更为复杂的目标可能会是“将实验室打扫干净, 并将所有的东西归位”。有时可能存在更复杂的偏好, 例如传递机器人要完成“邮件到达时就传送邮件, 有咖啡请求时要尽可能地服务, 但是传送标记为重要的信息更重要, 并且当克里斯请求咖啡时, 需要更快地为她服务”。
- 可能会有感知不确定性, 因为基于 Agent 有限的传感器, 它可能不知道世界中存在什么。
- 可能会有行为效用的不确定性, 无论是车轮滑动这样的低层次行为, 还是像 Agent

在传送咖啡给克里斯时不知道是否已经将咖啡放在她的书桌上这样的高层次行为，都具有不确定性。

- 可能存在多个机器人，它们可能相互协调来传递咖啡与包裹，也可能相互争夺电源插座。也可能会有儿童来戏弄机器人。
- 机器人有许多要学习的，例如地板的光滑程度与它的发亮程度成一定的函数关系，克里斯在同一天不同时段去哪里闲逛，克里斯会在什么时候要一杯咖啡，什么动作会得到最高的奖励。

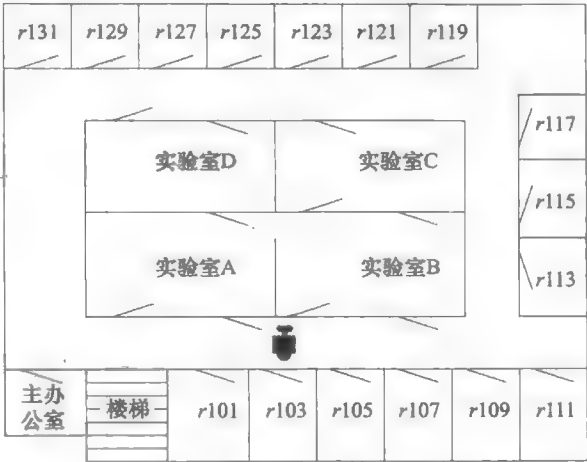


图 1-7 传送机器人环境图，说明一个典型的实验室环境，也说明了门的位置和它们开放哪一条路径

图 1-7 为传送机器人描述了一个典型的实验室环境。环境由 4 个实验室和很多办公室组成，机器人仅仅能够推开门，而且图中门的方向反映了机器人的移动方向。房间需要钥匙，钥匙可以从多种来源获得。机器人需要在这些房间之间传递包裹、饮料或菜肴。环境同样包括对机器人有潜在危险的楼梯。

32

1.6.2 诊断助手

诊断助手在一些特定系统，如就医的病人、房屋中的电力系统和汽车等，给人们提供建议。诊断助手应该为潜在的故障或疾病提供建议，要进行什么检查，要开什么样的处方。为了给出这些建议，诊断助手需要一个系统模型，包括关于可能的原因的知识，有效检查的知识，有效治疗的知识，以及系统的观察值(经常称之为症状)。

为了能够使用，诊断助手必须提供一些附加值，让人们能够容易使用它，而不是变得比其价值更为麻烦。诊断助手与因特网相连，可以从全世界吸收专家意见，它的行为会以最新的研究为基础。然而，它必须能够判定所提出的诊断或行为是不是合适。人类应该是对计算机系统持怀疑态度，因为它不透明而且令人费解。当人类为他们所做的负责时，即使是基于计算机系统的建议，他们也应该为所提出的行为提供合理的判断。

就图 1-3Agent 的黑盒定义来说，诊断助手有以下输入：

- 先验知识，例如开关和灯泡如何正常工作，疾病或故障怎样显现出来，检查所提供的信息，以及修理或治疗后的效果。
- 先前经验，是以前案例的一系列数据，包括修理或治疗后的效果，故障或疾病的流行程度，这些疾病和故障的特征的普遍程度，检查的准确度。这些数据通常是关于类似的人工产品或病人的，而非实际需要诊断的那个。
- 修复设备的目标以及在多种选择中进行权衡，例如是修理还是替换不同的部件，病人是否希望活得更久些，那将意味着会遭受更多的疼痛。
- 设备或病人的症状观察值。

诊断助手的输出是一系列治疗方法或检查的建议，以及提出这些建议的理由。

【例 1-10】 图 1-8 描述了一个房屋中的电力分布系统。在这个房屋中，电能通过断路

33

器进入房屋，然后转到插座或通过电灯开关转到电灯处。例如，当有电进入房屋，断路器 cb_1 打开，开关 s_1 和 s_2 都合上去或合下来时，灯 l_1 是开着的。这是一类模型，正常情况下，房主都会将电源放置在屋内，在给定开关位置，哪些灯开着哪些灯关着的状况下，通过模型确定所出现的问题。这个诊断助手可以帮助房主或电工检查电力问题。

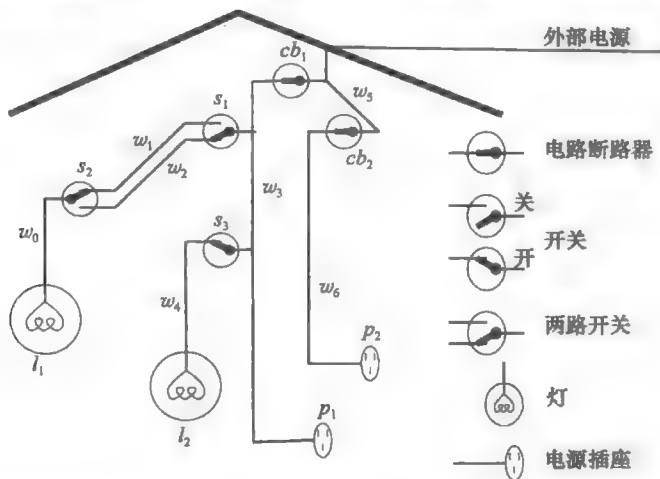


图 1-8 诊断助手的电气环境

每个维度都与诊断助手相关：

- 层次分解允许在处理低层原因的同时维持高层目标，而且能够进行详细的系统监控。例如在医疗领域，一个模块可以处理心脏监控的输出，并给出高层的观察值，如在心率发生变化时给出通知；另外一个模块会接收这个观察值和其他高层的观察值，并注意心率变化的同时引起的其他症状。在电气领域，图 1-8 描述的是一个抽象层次，更低层次可以详细描述电压、电线如何连接和开关的内部结构。
- 大多数系统太过复杂而无法用一系列的状态进行推理，所以它们经常用一系列的特征或个体部件及它们间的关系来描述。例如，人的身体可以用一系列各种各样的部件的特征值来描述。设计者在不知道实际个体的情况下，可能会想建立动力学模型。例如，电力诊断系统的设计者在知道哪些灯和开关存在于房屋里之前，也就是在知道它们的特征之前，为灯和开关如何工作建立模型。这时可以利用一系列的关系与交互，当知道个体时通过添加独立部件来完成这个模型。
- 对静态系统进行推理是可能的。例如，给定开关位置，当灯处于关闭状态时，对可能出现的问题进行推理。也可以对检查和治疗的结果进行推理，Agent 会一直进行检查和治疗，直到问题解决，或者 Agent 会实施持续的系统监控，持续修复任何问题。
- 感知不确定性是诊断需要面对的基本问题。如果 Agent 无法直接观察到系统的内部结构，这时就需要进行诊断。
- 效用不确定性也会存在，因为 Agent 可能不知道治疗结果，而且治疗经常会出现未预料到的结果。
- 目标，可能会像“确定出错的问题”这么简单，但通常会涉及花费、疼痛、预期寿命、诊断的正确率、疗效的不确定性、治疗的副作用不确定性等方面的复杂权衡。
- 尽管诊断通常是一个单 Agent 问题，但当涉及多个专家时，诊断会变得更复杂，

34

因为他们可能会有冲突的经验或模型，也可能会有多个病人来竞争 Agent 的资源（例如医生的时间、手术室）。

- 学习是诊断的基础。通过学习我们可以得知疾病的进展情况，以及治疗进行得怎么样。对于学习来说，诊断是一个具有挑战性的领域，因为所有的病人都是不同的，而每个医生的经验都只是针对带有任意特定特征集的少部分病人。医生们看到是有偏差的人口样本，那些来看病的人，通常都会有一些不寻常的或疼痛的病症。
- 诊断通常需要快速地回答，没有时间进行详尽的推理或做到完全合理。

1.6.3 智能指导系统

智能指导系统是在某些领域指导学生的计算机系统。

例如，在基础物理的指导系统中，如力学，系统会介绍理论知识，给出一些固有的例子，还会向学生提问，当然它必须能够理解学生的回答，并基于学生的回答判定他掌握知识的程度。这会影响到学生向系统提出什么问题以及系统向学生提出什么问题。学生向系统提问问题，系统应该能够解决在物理学领域内的问题。

35

就图 1-3 Agent 的黑盒定义来说，智能指导系统有以下输入：

- 先验知识，由 Agent 设计者提供，包括教学主题、教学策略、可能误差和学生的误解。
- 先前经验，是指导系统通过与学生交互获得的，关于学生犯的错误、学习一些知识需要多少例子和学生所遗忘的。所获得的信息可以针对一般学生，也可针对特定学生。
- 关于每个话题的重要性、学生期望得到的成就水平以及与可用性相关的花费的偏好，它们之间通常会有很复杂的权衡。
- 学生测验结果的观察值，学生与系统交互（或不交互）的观察值。学生可以提出问题或提供新的实例。

指导系统的输出是给学生提供的信息、学生应该要做的测试、问题的答案给父母和教师的报告。

每个维度都与指导系统相关：

- 应当有 Agent 的层次分解和教学任务分解两个部分。在学习高层概念之前，学生应该被教授基础知识。指导系统有高层教学策略，但是在更多较低的层次上，它必须为一个测验设计实例的细节部分与特殊问题。
- 指导系统可以用一系列学生的状态来进行推理。然而，实际上学生以及教学科目一般都具有多个特征。如果所有的例子都是固定的，而且是针对一个学生进行推理，那么物理指导就能够用在设计时就已知的一系列特征来进行推理。对于更复杂的情况，指导系统应该会涉及个体及其之间的关系。如果指导系统或学生能够利用多个个体创造实例，那么这个系统在设计时可以不知道特征，并用一系列个体及其之间的关系进行推理。
- 在规划期方面，在测验的整段时期内，认为此领域是静态的，而且学生在测验时不会进行学习的假设是合理的。对一些子任务来说，有限期比较合适。例如，可能会有教学、测验、再教学这样的序列。在其他情况下，在设计时系统根本不知道会进行到哪一步，直到学生掌握一些概念，这是一个无限期。还可以对教学建

模以作为一种持续的学习和测验过程,并伴随着适当的休息,同时不希望系统在此过程中终止。

36

- 不确定性起很大的作用。系统不能直接观察到学生的知识,它只是基于学生提问的问题或没有提问的问题以及测验结果的一些感知输入,系统不会确切地知道某个特定的教学片段的确定效果。
- 尽管系统有一些像教授某些特定概念这样的简单目标,但它可能更需要考虑很多复杂的偏好。其中一个理由是由于不确定性,根本就没有办法保证学生能够知道所教授的概念;任何使学生知道一个概念的可能性最大化的方法都很恼人;因为当疲劳或无聊导致学生的微小错误时,系统都会持续反复地进行教学与测验。更多复杂的偏好使如下方面的权衡成为可能:概念的充分讲解、学生的厌烦情绪、所花的时间和重复测验的数量。用户可能对教学风格有偏好也应该考虑。
- 将系统作为一个单 Agent 问题进行处理更为合适。然而,学生、老师和父母都有必须考虑的不同偏好。每一个 Agent 都能不说出真相地策略性地实施动作。
- 我们期望系统能够学习以下内容:教学策略是什么,测验概念时的一些问题怎么样,学生犯的共同错误是什么。它能够学习一般知识或针对某一主题的特定知识(例如,对于教授力学知识来说,学习什么策略适用),或者针对某个特定学生的知识,如对 Sam 管用的知识。
- 选择最合适的材料来介绍知识需要花费大量的计算时间。然而,学生必须及时做出反应,有限理性能够确保在学生等待时系统不会计算过长时间。

1.6.4 交易 Agent

交易 Agent 就像是一个机器人,但它不是与物理环境进行交互,而是与信息环境进行交互。它的任务是为用户获取商品及服务。它必须了解用户的需求,并与卖者进行交互(如在网上)。最简单的交易 Agent 是拍卖场所为用户进行代理投标的 Agent,系统会在达到用户的价格限制之前一直维持出价的状态。更复杂的交易 Agent 会购买多个互补的物品,像预订机票、宾馆、出租车这样一个组合,并权衡竞争性偏好。另外一个交易 Agent 的实例是一种可以显示家里有多少食物与杂货的 Agent,可以监控价格,并在购买之前调整顺序,使费用降低到最小。

在图 1-3 Agent 的黑盒定义中,交易 Agent 有以下输入:

37

- 关于货物或服务类型、出售业务、拍卖流程等的先验知识;
- 先前经验,哪儿是寻找特价物品的最好地方,拍卖中价格会随时间如何变化,特价什么时候会有上升的倾向;
- 用户想要的物品的偏好,怎样权衡有冲突的目标;
- 什么物品是可用的、物品的价格、可购买的时间范围这几个方面的观察值。

交易 Agent 的输出是用户能够接受或拒绝的建议,或者是一个实际购买。

交易 Agent 应该考虑以下维度:

- 由于域的复杂度,层次分解是必要的。考虑为旅行者做好假期所有的安排与购物这样一个问题。如果有一个专门购买机票并且能优化转乘和时间的模块,而不是在做这些事情同时来决定通过哪个门到出租车站,就会更简单些。
- 交易 Agent 的状态空间对于个体状态推理来说太大了。还存在太多的个体,因而无法用特征的方式进行推理。交易 Agent 不得不按照如顾客、天数、宾馆、航班

等这样的个体进行推理。

- 交易 Agent 一般不会只进行一次购物，通常会进行一个序列的购物，会有大量的序列决策(例如，预订宾馆客房可能会需要预订陆地交通，然后是存储包裹)，通常会为持续的采购制定计划，例如 Agent 必须确保家里一直会有足够的食物。
- 感知不确定性，因为交易 Agent 不会知道所有的可用选择以及它们的可用性，但是必须找出那些很快会过时的信息(例如，是否有宾馆会被预订一空)。旅行 Agent 不会知道一个航班是否会被取消或延迟、旅客的行李是否会丢失等信息。这种不确定性意味着 Agent 必须为意料之外的事情做计划。
- 效用不确定性，因为 Agent 不知道一次尝试性的购物会不会成功。
- 复杂的偏好是交易 Agent 的核心部分。主要问题是允许用户描述他想要的东西。用户的偏好一般是功能描述，而不是部件描述。例如，典型的计算机买主一般不知道要买什么样的硬件，但是知道他们需要什么样的功能，以及能够使用那些也许还不存在的新特性的灵活性。类似的，在旅游领域，用户想要什么样的活动依赖于具体的地理位置。即使他们可能根本不知道这些风俗是什么，用户也可能想在目的地参加当地的风俗活动。
- 交易 Agent 必须对其他的 Agent 进行推理。在贸易中，价格由供需关系决定；这就意味着对其他竞争性 Agent 进行推理是重要的，比如在很多物品通过拍卖进行出售的世界中。当物品必须互补时，推理变得很困难，如航班与宾馆预订，还有可以相互取代的物品，比如公共汽车或出租车。
- 交易 Agent 应该了解一些信息，例如，什么物品卖得比较快，哪个供应商可靠，在哪里能有比较划算的交易，会有什么意料之外的事发生等。
- 交易 Agent 面临严峻的通信局限性。当发现物品可用并与其他物品协调时，物品可能已经卖完了。这种情况会在卖家同意保有某些物品(同时不会卖给别人)时有所缓解，但卖家不可能在别人也想买的时候长时间持有这些物品。

38

由于交易 Agent 的个性化本质，它应该会比通用购买者做得更好些，比如，只提供套装旅游。

1.7 本书概述

本书的余下部分对由复杂性维度所定义的设计空间进行探索，对每个维度进行独立的合理考虑。

第2章主要分析图1-3中所提到的黑盒的内部结构，并讨论 Agent 的模块及层次分解。

第3章主要讨论决策未来行为的最简单情况，单个 Agent 用显式状态推理，没有不确定性，有要完成的目标，但是存在不确定期。在这种情况下，解决目标这个问题可以抽象成图中的路径搜索问题，并介绍了如何利用本领域的额外知识进行搜索。

第4、5章主要介绍如何利用特征。具体地说，第4章主要考虑怎样在给定的约束条件下找到可能的状态，这个约束是以变量形式表示的特征的值的配置。第5章阐述在所有状态都满足给定的约束集时，如何确定命题是否为真。

第6章主要讲述如何用不确定性进行推理。

第7章介绍如何从先前经验及数据中学习。它包括学习中最常见的情况，即利用特征的监督学习，从中可以学到被观察目标的特征集合。

第 8 章考虑规划问题, 具体对状态及动作的基于特征的表达进行表示与推理。第 9 章介绍不确定性中的规划问题, 第 10 章将这种状况扩展至多个 Agent。

39 第 11 章介绍不确定性下的学习及强化学习。

第 12 章介绍如何用个体及关系进行推理; 第 13 章主要讨论的是本体, 以及如何建立基于知识的系统; 第 14 章说明个体及关系的推理如何与规划、学习及概率推理相结合。

第 15 章回顾人工智能的设计空间并说明本书提供的材料如何适应设计空间。同时也介绍一些关于建立智能系统的伦理思考。

1.8 本章小结

- 人工智能是对智能行动的计算 Agent 的研究。
- Agent 在环境中动作, 它只能访问自己的先验知识、历史观察值、目标及偏好。
- Agent 是一个通过操纵符号来决定要做什么的物理符号系统。
- Agent 的设计者应该考虑模块性、如何描述这个世界、向前计划多远、感知和行为效果的不确定性、目标或偏好的结构、其他的 Agent、如何从经验中学习和所有真实 Agent 都只有有限的计算资源的事实。
- 为了通过计算机解决问题, 计算机必须具有有效的表达方式以便于推理。
- 为了知道什么时候已经解决了问题, Agent 必须对适当解的构成做一个定义, 例如, 是否它必须最优, 或近似最优, 或几乎总是最优, 或者一个可满足解是否适当。
- 在选择表示法时, 应该用那种尽可能接近问题的表示法, 这样就可以很容易地确定所表示的内容, 检查它的正确性并能够维护。通常用户都会想要一个解释, 为什么他们应该相信答案。

1.9 参考文献及进一步阅读

本章思想来源于很多资料。在这里, 我们向那些有贡献的作者致以诚挚谢意。而其他的大部分想法来自民间人工智能方面的研究, 我们无法将贡献仅归功于任何人。

Haugeland[1997]有一批关于人工智能哲学基础的论文, 其中就包括提出图灵测试的 Turing[1950]的经典论文。Cohen[2005]对图灵测试做了最新讨论。

40

Nilsson[2009]对人工智能的历史作了详细描述。Chrisley 和 Begeer[2000]发表了很多关于人工智能的经典文章。

物理符号系统假说是由 Newell 和 Simon[1976]提出来的。参考 Simon[1996], 他讨论的是在多学科背景下符号系统的作用。Haugeland[1985]讨论了自然智能、合成智能与人工智能的区别, 也给出了在解释型自动形式符号系统以及 Church-Turing 理论等方面有用的介绍材料。Brooks[1990]和 Winograd[1990]对符号系统假说做了批判。Nilsson[2007]利用最近的一些评论对这个假说做了评价。

任意时间算法的使用应该归功于 Horvitz[1989]、Boddy 和 Dean[1994]。Dean 和 Wellman[1991, 第 8 章]、Zilberstein[1996]和 Russell[1997]对有限理性做了介绍。

Kirsh[1991a]、Bobrow[1993]讨论了人工智能的基础以及人工智能的研究范围, 还有相应卷中的论文, 以及 Schank[1990]与 Simon[1995]。Lenat 和 Feigenbaum[1991]与 Smith[1991]讨论了人工智能中知识的重要性。

Gardner[1985]、Posner[1989]、Stillings、Feinstein、Garfield、Rissland、Rosenbaum、Weisler 和 Baker-Ward[1987]对认知科学做了概述, 阐述了人工智能以及其他一些学科在这个领域中扮演的角色。

购物 Agent 能够变得非常复杂。Sandholm[2007]描述了 Agent 如何用复杂偏好采购多种产品。

对于本书来说, 很多人工智能课本很有价值, 可以作为参考书目成为本书的补充, 为人工智能提供了不同的视角。尤其是 Russell 和 Norvig[2010]给出了一个更为广泛的人工智能概述, 并为本书中的很多主题提供了补充资料, 他们提供了科技文献的一个优秀评述, 对此本书不打算重复。

人工智能百科全书[Shapiro, 1992]是由本领域的领军人物编写的关于人工智能的百科词典, 为一些

经典主题提供了背景,其中也收藏了大量经典研究论文。本书读者最感兴趣的一般馆藏是 Webber 和 Nilsson[1981]与 Brachman 和 Levesque[1985],更多专业馆藏会在相应的章节中给出。

国际人工智能促进协会(AAAI,其前身是美国人工智能协会)通过他们的 AI 主题网站(<http://www.aaai.org/AITopics/html/welcome.html>)提供了很多基础资料与新闻。由 AAAI 出版的《AI 杂志》有很多优秀的概述性文章以及特定应用领域的描述。IEEE 智能系统期刊(IEEE Intelligent Systems)也提供了很多关于人工智能研究方面的可获取的文章。

有很多学术期刊提供了更深入的研究,还有一些能够提供最新研究的会议。这些期刊包括《人工智能》(Artificial Intelligence)、《人工智能研究》(Journal of Artificial Intelligence Research)、《IEEE 模式分析与机器智能会刊》(IEEE Transactions on Pattern Analysis and Machine Intelligence)和《计算智能》(Computational Intelligence),还有一些专业期刊,如《神经计算》(Neural Computation)、《计算语言学》(Computational Linguistics)、《机器学习》(Machine Learning)、《自动推理》(Journal of Automated Reasoning)、《近似推理》(Journal of Approximate Reasoning)、《IEEE 机器人与自动化会刊》(IEEE Transactions on Robotics and Automation)和《逻辑编程的理论与实践》(Theory and Practice of Logic Programming)。大多数前沿研究首先出现在会议上。一般人最感兴趣的是两年一度的人工智能国际联合大会(IJCAI)、AAAI 年会、欧洲人工智能会议(ECAI)、泛太平洋人工智能国际会议(PRICAI)、各种各样的国家级会议,以及很多专业会议及研讨会。

41

1.10 习题

1.1 为下面每一项给出 5 个原因:

- (a) 一条狗比一只蠕虫更为智能。
- (b) 一个人比一条狗更智能。
- (c) 一个组织比单个的人更智能。

基于上述讨论,给出“更智能”的含义。

1.2 尽可能多地举出一些研究某种智能行为的学科。找出研究的是智能行为的哪一方面,用什么工具来研究的。尽可能广泛地去理解有关智能行为的定义。

1.3 说出 AI 的两种应用(不是应用类别,是具体的应用),每个应用最多用一页文字来描述。你可以从以下几个问题考虑:

- (a) 应用具体是做什么的(如控制一台航天器,诊断一台复印机,为计算机用户提供智能帮助)?
- (b) 使用了什么人工智能技术(如基于模型的诊断,信念网络,语义网络,启发式搜索,约束满足)?
- (c) 它的表现如何?(如根据作者还是独立的评论者?它跟人类相比怎么样?作者怎么知道它的表现怎么样?)
- (d) 它是个实验系统还是装备系统?(它有多少用户?这些用户需要什么样的专业知识?)
- (e) 它为什么是智能的?什么方面使它成为了一个智能系统?
- (f) [可选]使用什么编程语言及环境完成的?使用哪种用户界面?
- (g) 参考文献:你从哪里得到关于这个应用的信息?其他人应该到什么样的书中、文章中或网站上去了解这个应用?

1.4 选择本书中没有涉及的 4 对维度。对于每一对,试给出一个维度交互的常识性例子。

42

Agent 体系结构和分层控制

所谓分层或分层系统，是指一个系统由多个相互关联的子系统组成，而每个子系统又可再次分层直至到达最底层基本子系统为止。自然界的大部分系统中，何时停止分层及什么是基本元素的随意性极大。物理学中会经常使用基本粒子的概念，但是粒子早已证实不是物质的最底层基本元素。

我们从实验观察到自然中的大部分复杂系统都展示出分层结构。从理论上讲，我们希望世界上的复杂系统都具备分层结构，其复杂系统可从简单系统演变而来。

——Herbert A. Simon[1996]

本章主要讨论智能 Agent 在实时环境中如何感知、推理和动作。实际上，这涉及 Agent 的内部结构。正如上面西蒙所指出的，层次分解是复杂系统(如智能 Agent)设计中的重要部分。因此本章主要描述了用层次分解方式设计智能 Agent 的方法、构建智能 Agent 的方法以及构建“智能”所需的知识。

2.1 Agent

Agent 是指在具体环境中具有动作能力的对象，如一个人、一个机器人、一条狗、一只小虫子、一阵风、重力、一盏灯或者一个能购物和售货的计算机程序。

有意图的 Agent 是具有偏好的，它会偏爱于某些环境状态，并将采取行动来达到它们最偏好的状态。无意图的 Agent 则会聚集在一起并被称为自然。建模时需设定一个 Agent 是否为有意图 Agent，此建模假定是否合适要根据环境来定，如在一些情境中，需要将某条狗定义为有意图的，而有时候则不需要。

如果一个 Agent 不具有偏好，由定义可知它不会关心其所处的环境状况，因此也不会关心自己做什么。设计此类 Agent 的唯一目的是逐步培养其偏好，使其更偏爱某种环境状态并尽力实现之。Agent 并非必须知道自己的偏好，如一个恒温器，它只需感知环境，并控制加热器开和关即可。它嵌入了偏好，即让房间中的人员处在一个合适的温度中，尽管它并不知道具有此偏好。Agent 的偏好一般是 Agent 设计者的偏好，但是有时在 Agent 运行中也可赋予目标和偏好。

Agent 需要一个“主体”来与环境进行交互。一个嵌入式的 Agent 有一个物理的主体。机器人就是一个人造的具有目的性的嵌入式 Agent。有些 Agent 则仅在信息空间存在，也被称为机器人，这里我们说的 Agent 指的便是此类。

本章讨论了如何构建有意图的 Agent，我们用机器人作为主要的实例，因为机器人学科中已有相当多的研究，且许多术语来自于机器人学科。当然，此讨论适用于所有 Agent。

Agent 通过传感器来接收信息。一个 Agent 的动作取决于通过传感器获取的信息。这些传感器有可能反映环境的真实状态。传感器可能是有噪声的、不可靠的、坏掉的，甚至有时传感器是可靠的，但是它传回的关于环境的信息却是模棱两可的，而 Agent 却必须依靠其获取的信息来行动。这些信息经常很弱，例如，传感器 s 看上去会生成数值 v 。

Agent 通过它们的**执行器**(也叫 effectors)来动作。执行器同样可能是有噪声的、不可靠的、行动缓慢的或者坏掉的。Agent 所控制的是其发送给执行器的信息(命令)。Agent 经常会采取行动去寻找更多的环境信息,如打开橱柜门查看物件或者对学生进行测试来测定他们的知识。

2.2 Agent 系统

图 2-1 展示了 Agent 与环境的一般交互过程,该图所示整体即是我们所说的 Agent 系统。

一个 Agent 系统是由 Agent 和其所在环境构成。Agent 接收环境中的刺激,然后做出相应动作。

一个 Agent 由**主体**(body)和**控制器**(controller)两部分组成。控制器从主体处接收感知,然后将命令送至主体处。

主体包括**传感器**和**执行器**,传感器将外部刺激转化为感知,执行器能将命令转换成动作。

刺激包括光、声音、键盘上输入的单词、鼠标移动或者物理冲击,也包括从网页或者数据库中获取的信息。

常见的传感器包括触摸传感器、相机、红外传感器、声呐、麦克风、键盘、鼠标或者通过网页来抽取信息的 XML 阅读器。作为原型传感器,相机感知进入它的光束,将其转换成亮度数值的二维数组,称为**像素**。有时会使用多维像素数组来表示不同的颜色或者满足多镜头相机的需求。这些像素数组可被控制器感知。而更多的时候,感知对象通常有着更高层的特征,如线、边或深层次的信息。通常来说,接收的都是特定信息,如明亮的橙色圆点的位置,学生关注的演出部分或者人们打出的手势信息。

动作一般包括转向、加速、移动关节、讲话、展示信息或者向某一网址发送邮件的命令。命令又包括低级命令(如将发动机的电压设定为某个数值)和高级命令(如令一个机器人进行某些运动,例如“停止”,“以 1m/s 的速度向正东方向运动”或者“到 103 房间去”)。执行器同传感器类似,都包含噪声。例如,停止是需要时间的;机器人在物理规则下运动,所以具备动量,且信息传递需要时间。机器人也许最终只是以接近 1m/s 的速度运动,接近正东方向,且速度和方向都是不断波动的,甚至运动到某一指定房间的行为可能会由于各种原因而失败。

控制器是 Agent 的大脑,本章的剩余部分将主要讨论如何构建一个控制器。

Agent 的功能

Agent 应构建为实时性的:应能即时接收传感器信息,并即时做出反应。特定时刻的 Agent 动作是其输入的函数。我们首先考虑时间的概念。

设 T 为时间点集合,假定 T 有序,且任意两个时间点的时序距离均可测定。大体上,我们设定 T 可以映射到实线上的一些子集。

若任意两个时间点间仅存在有限个时间点,则称 T 是**离散的**,如每一天或 1/100 秒有一个时间点,或者有趣事件发生的时刻也可以是时间点。但如果 T 中任意两个时间点间都存在另一个时间点,则认为 T 是**稠密的**,这表明 T 中的任意两个时间点间存在无限多个时间点。离散时间存在如下特征:除了最后时间点,其他任何时间点均存在下一时间点。

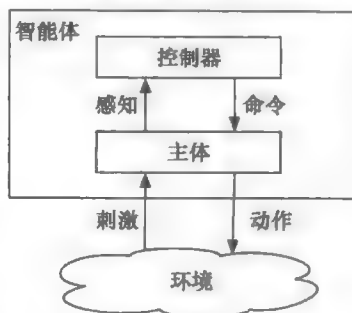


图 2-1 一个 Agent 系统及其构成部分

而稠密时间则无“下一时间点”。我们将时间初始化为离散的, 且没有尽头。因此每个时间点都存在下一时间点。我们令 $t+1$ 为时间点 t 的下一时间点, 但这并不代表时间点之间是等间隔的。

假定 T 有初始时间点, 我们这里称其为 0。

假定 P 为所有可能感知对象的集合。一个感知轨迹或者一个感知流则为一个从 T 到 P 的函数, 它描述了每一时间点所观察到的事物。

假定 C 是所有命令的集合。一个命令轨迹为从 T 到 C 的函数, 其代表在每个时间点的命令。

【例 2-1】 我们来考虑一个家居交易 Agent, 它监控着多种家居用品价格(如它监视着某些特定交易, 并记录卫生纸的涨价情况)及家中相应存量。它必须决定是否购进某物及在何种价格时购进。感知对象为当前商品价格及家中存量。命令为 Agent 决定购买的各类商品的数量(若不购进则数目为 0)。感知轨迹描述了每个时间点(如每天)的商品价格和库存数量, 如图 2-2 所示。命令轨迹描述每个时间点 Agent 购进的商品数量, 如图 2-3 所示。

实际购入动作依赖于命令, 但有可能有所不同。如 Agent 发出一条以某一特定价格购入 12 卷(1 打)卫生纸的命令, 但并不意味着 Agent 实际购入了 12 卷卫生纸, 因为可能由于网络通信问题, 仓库中的卫生纸已卖完, 或者在决定买和实际买之间时价格已经发生变动。

一个 Agent 的感知轨迹是控制器在过去、现在和将来接收到的所有感知信息的序列。命令轨迹则是控制器在过去、现在和将来发出的所有命令信息的序列。命令

可以是历史感知的函数。这就产生了转换的概念, 从感知轨迹映射到命令轨迹的函数。

因为所有的 Agent 都处于时间流中, 所以 Agent 不能真正地观察到全部感知序列; 在任意时刻, 它只能观察到截至现在的感知轨迹。在 $t(t \in T)$ 时刻只能观察到 t 时刻及其之前的信息流。其命令只能根据其经验来决定。

转换过程是有因果联系的, 如果对于所有时刻 t , 在 t 时间的命令都仅由 t 和其之前的感知信息决定。因果限制是必需的, 因为 Agent 处于时间流中, 所以 t 时刻的命令不可能依靠 t 之后的感知信息。

控制器完成因果转换的具体实现。

Agent 在时间 t 的历史包括其在 t 时刻和之前的感知流和在 t 时刻和之前的命令流。

因此, 一个因果转换可看做是从 Agent 在 t 时刻的历史到其在 t 时刻发出的命令的函

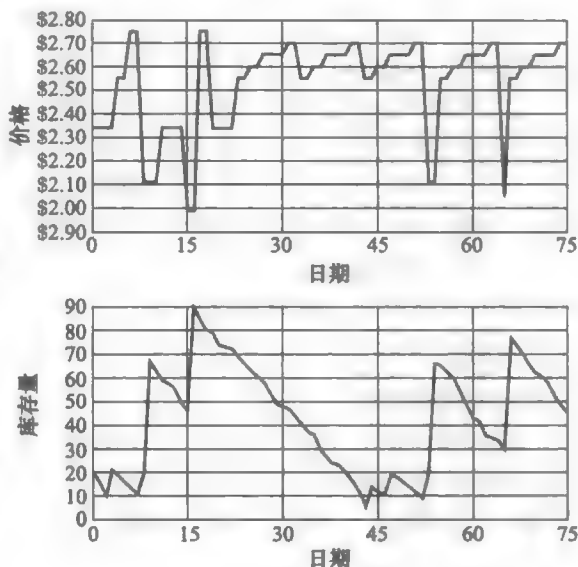


图 2-2 例 2-1 的感知轨迹

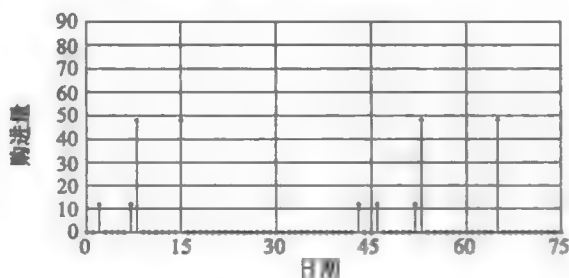


图 2-3 例 2-1 的命令轨迹

数。这可被认为是 Agent 的最规范描述。

【例 2-2】 继续例 2-1 的情况，一个因果转换可认为，对于任意时刻，Agent 需要购买多少日用品取决于历史价格、历史库存量(包括时下价格和现有库存)及过往购买历史。

因果转换的示例如下：当库存量低于 5 打且价格低于过去 20 天的平均价格的 90% 时则购买 4 打；若库存量低于 1 打时购进 1 打；其他情况下不购进。

尽管因果转换是用户历史的函数，但其不能直接实现，因为 Agent 不能直接获得它们的全部历史信息，它只能获得当前的感知信息和它仍能记住的信息。

一个 Agent 在时间 t 时的信念状态是其所能记住的所有以前时间的信息的总和。Agent 只能获取其存在信念状态中的历史。因此信念状态蕴含了其现在和将来命令所能用到的所有历史信息。在任何时间，Agent 都能访问其信念状态和感知信息。

信念状态可以包含任何信息，仅受制于其存储器大小和处理能力限制。这是信念的一个非常普通的定义；有时我们会使用信念的一个更特别的定义，如 Agent 的信念是关于世界上哪些为真，或是关于环境的动态改变，或是关于其在未来会做什么。

一些信念状态的实例如下所示：

- 有固定顺序指令队列的 Agent 的信念状态可能是一个程序计数器，记录着该序列的当前位置。
- 信念状态可能包含有用的特定事实，例如，传送机器人在何处放下包裹去找钥匙，或者它找钥匙时已经去过的地方，记住任何当前无法立即观察却相对稳定的信息对 Agent 是有用的。
- 信念状态可以编码成一个模型或者整个世界状态的部分模型。Agent 可以保留其对于当前世界状态的最佳猜测，也可以是可能世界状态的一个概率分布，详情内容见 5.6 节和第 6 章。
- 信念状态可以是世界的动态表示、感知的意义，Agent 可以使用其感知来判断世界中什么是正确的。
- 信念状态可以编码为 Agent 的愿望、它需实现的目标、它关于世界的信念、它的意图或者为了实现目标而准备实施的步骤。当 Agent 行动和观察世界时，这些能被维持，例如，移除已经达成的目标或当发现更适合的步骤时改变意图。

控制器必须保存 Agent 的信念状态并决定每个时刻发出何种命令。当它做这些时，它应该获得的信息需包括自身信念状态和当前感知。

离散时间下的信念状态转换函数可表示为

$$remember: S \times P \rightarrow S$$

其中， S 是信念状态集， P 是可能认知的集合； $s_{i+1} = remember(s_i, p_i)$ 表示状态 s_{i+1} 是在信念状态 s_i 之后观察到 p_i 得出的信念状态。

指令函数可表示为

$$do: S \times P \rightarrow C$$

其中， S 是信念状态集， P 是可能认知的集合， C 是可能指令的集合； $c_i = do(s_i, p_i)$ 表示当前的信念状态为 s_i ，观察到 p_i 时控制器需要发出的指令 c_i 。

信念状态转换函数和指令函数一起描述了 Agent 的因果转换过程。可以发现，因果转换是 Agent 历史的函数，Agent 不必访问历史，但指令函数是 Agent 信念状态和认知的函数，这两个是 Agent 必须要访问的。

【例 2-3】 为了实现例 2-2 中的因果转换，控制器必须跟踪过去 20 天内的价格。通过使用均值(*ave*)的跟踪数值，可以更新均值

$$ave := ave + (new - old) / 20$$

其中，*new* 是最新时刻的价格，*old* 指记住的最早时刻的价格，它在使用后立即会被丢弃。其中最初 20 天的数据将会做一些特殊处理。

为了使控制器更加简单，其不需记忆过去 20 天的历史来获取平均值，而改为仅存储平均值，并使用平均值来替代最早价格。信念状态便仅包含一个数据(*ave*)。更新平均值的状态转换函数为

$$ave := ave + (new - ave) / 20$$

这个控制器很容易实现，对于在过去 20 个时间单位之前发生的变化不敏感。这种保留平均估计值的方法是强化学习中的时间差分法的基础。

如果仅存在有限的可能信念状态，控制器可被称为有限状态控制器或者有限状态机。因素化表达是指信念状态、感知和命令均由特征来定义。如果存在有限特征，且每个特征仅存在有限种可能的取值，那控制器就是一个因素化有限状态机。更全面的控制器可以采用无限数量的特征或特征取值来构建。一个有可数状态的控制器可以计算任何图灵机可计算的问题。

2.3 分层控制

在图 2-1 中讲述的一种构建 Agent 的方法是将主体分为两部分：传感器和一个复杂的感知系统，感知系统将对世界的描述输入给推理引擎来实现控制器，然后能够生成传给执行器的命令。但这对智能系统而言是一个很差的结构，因为它太慢了，使得其很难缓解以下两个矛盾：复杂问题的缓慢推理和 Agent 所需的高级目标的快速反应，例如，躲避障碍。它也不清楚，在其可操纵范围之外存在一个独立的世界(见习题 2.1)。

另一种可行结构是在图 2-4 中描述的控制器的分层。每一层都将其下层看做一个可从中获取感知及送出命令的虚拟主体。越低层，反应速度越快，用于那些需要快速反应的情况，且能将对环境更简单直白的感知传给高层，隐藏不重要的信息。

总的来说，不同时间的状态之间，会有大量的特征在层与层之间传递。

对于每个时刻的每一层存在如下三类输入：

- 来自于信念状态的特征，指的这些特征所记住的或以前的值。
- 从下层结构得到的感知的特征。
- 从上层结构得到的命令的特征。

对于每个时刻的每一层存在如下三类输出：

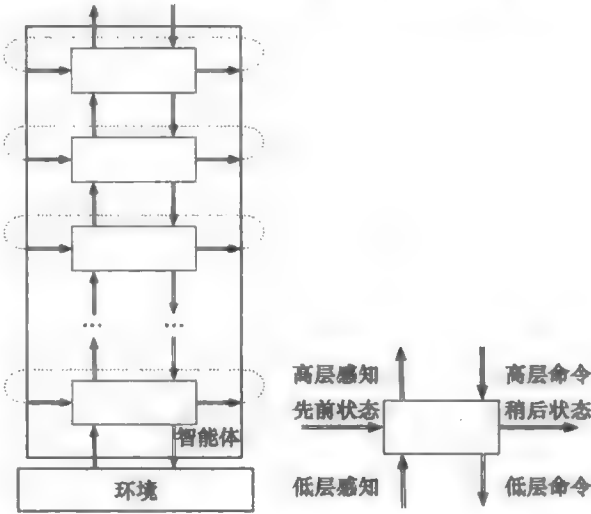


图 2-4 一个理想的分层 Agent 系统结构。未标注的矩形表示层，双线表示信息流。虚线说明某一时刻的输出作为下一时刻的输入

- 对应上层的更高等级的感知。
- 对应下层的低等级命令。
- 信念状态特征的下一个值。

层级的实现方式决定了层的输出是如何由输入的函数得出。此函数的计算可以涉及任意计算，但目的是使每层尽可能简单。

为了实现控制器，层上的任何输入必须有确切的赋值来源。每个感知或者命令输入应该源自与其相连的其他层的输出。其他的输入来源于记住的信念。而层上的输出不必有相连对象，或者可与多个输入相连接。

高级推理在高层上给出，一般是离散的、定性的，而底层中给出的低级推理则是连续和定量的(见下面“定性与定量表示”的内容)。一个可推理产生离散和连续两种值的控制器被称为混合系统。

51

定性与定量表示

很多的科学工程都会考虑使用微积分作为主要工具来进行数值量化的**定量推理**。而**定性推理**一般使用逻辑来在给定参数条件下进行性质上的区分而非数值上的推理。

定性推理很重要，原因如下：

- 一个 Agent 可能无法获知确切的数值。例如，对于一个倒咖啡的传送机器人，它也许不能计算出倒咖啡时咖啡壶的最佳倾斜角度，但是一个简单的控制规则就足够给杯子加入合适高度的咖啡。
- 无视数量的推理有时可能更为合适。例如，你也许希望机器人的策略是，只要能在其可正常操作的范围内，在工作时便可无视一些东西，如机器人负载是多少、地板有多滑或者电池的实际电量还有多少。
- 一个 Agent 需要做定性推理来决定哪条定量规则更为合适。比如，如果传送机器人正在向咖啡杯中加咖啡，不同的定量规则适合于决定在以下情况下咖啡往哪里流：当咖啡壶的倾斜角度不够咖啡流出时，当咖啡加入至一个没满的杯子时，当咖啡杯满并流出时。

定性推理采用离散值，其中有多种不同的形式：

- **标志**是用于在建模个体中进行定性区分的数值。在咖啡这个例子中，有很多重要的定性区分，包括咖啡杯是否为空、部分满、满了。这些标志值对用于预计如果杯子足够倾斜或者咖啡倒入杯中时会发生什么来说是必需的。
- **数量级顺序推理**是种无视轻微差别的近似推理。举个例子，一个部分满的咖啡杯可能已经满足传送要求，还有一半满和接近空。这些**模糊定义**并没有一个明确定义的边界。在杯中咖啡的实际量间和定性描述存在某些关系，但是其中并没有严格的数值因子。
- **定性的导数**可以表明是否一些数值在增加、减少或是不变。

一个灵活的 Agent 在其做定量推理之前需要做定性推理。有时只需要定性推理。因此 Agent 并不总是需要进行定量推理，但是有时定量和定性推理都需要。

52

【例 2-4】 我们认定一个传送机器人能在躲避障碍物的同时执行高级的导航任务。假定传送机器人需要在避开可能的障碍物的同时按照顺序访问图 1-7 中环境的一系列位置。

假设传送机器人有像轿车一样的轮子，且每个时刻可以选择直走、右拐或左拐。但不

能停止运动。速度是恒定的，且唯一的指令是设定转向角度。轮子的转向是瞬时完成的，但是调整至一个确定角度需要时间。因此机器人只能向正前方运动或按照一个固定的半径绕圈。

机器人有一个位置传感器，可以给自己提供当前坐标和方向。还有唯一一个伸向正前稍偏右的触觉传感器，用于探测是否碰触到障碍。在下面的例子中，传感器指向前方偏右 30 度的方向。机器人不自带地图，且环境可以改变(如：障碍可以移动)。

对于此类传送机器人的分层控制器已在图 2-5 中给出。此机器人有个高级规划需要执行。这个规划是按顺序访问一系列地点。机器人需要感知环境，然后在环境中移动以便执行规划。底层的具体细节并未在图中给出。

最高层，被称为后续规划层，在例 2-6 中有描述。那一层获取一个用于执行的规划。此规划是所需访问地点的字的顺序排列。地点会按照顺序选中，每次选中的地点都会转换成当前目标。此层将决定目标的 $x-y$ 坐标位置。这些坐标是提供给底层的目标位置。高层知道位置的名字，而底层仅知道坐标。

高层中保留着相关的信念状态，包含一系列机器人仍需访问的地点的名称和当前目标的坐标。它以当前目标坐标的形式给中间层传达命令。

中间层，也可被称为前往目标并躲避障碍层，它试图在避开障碍的同时前往当前目标地点。中间层在例 2-5 中有描述。目标的位置，即 *target_pos* 可从上层获得。当中间层到达目标位置时，它会通过将 *arrived* 设为真来告诉顶层其已完成目标。中间层可给一直在等待的顶层发送中断通知来达成此目的，当然顶层也可能一直在监听中间层的运行情况来判定何时 *arrived* 状态为真。当 *arrived* 变为真时，顶层接着会将目标位置改为规划中下个位置的坐标点。因为顶层改变了当前目标位置，中间层必须使用前一个目标位置来决定其是否到达目标。因而，中间层必须从顶层获得当前和之前的目标坐标：之前的坐标位置用于决定其是否已经到达，而现在的目标坐标是用于决定将往何处去。

中间层可以获知机器人的当前坐标和方向，且能决定其触觉传感器是开启还是关闭。它使用一个非常简单的策略，即直行向目标直至被阻碍，此时其会选择左拐。

中间层是建立在底层之上的，底层会提供机器人的一些简单看法。底层可称为行进机器人和报告障碍及位置层。它执行转向命令并报告机器人的位置、方向及传感器的开关状态。

在层的内部是特征，而特征可以为其他特征值和该层输入的函数。每个特征有一个所依赖的特征或输入的一个弧形连接。特征的相互依赖图一定是非循环的。非循环图使得控制器可以通过运行一个顺序指定数值的程序而实现。而构成信念状态的特征可从内存中读取和写入。

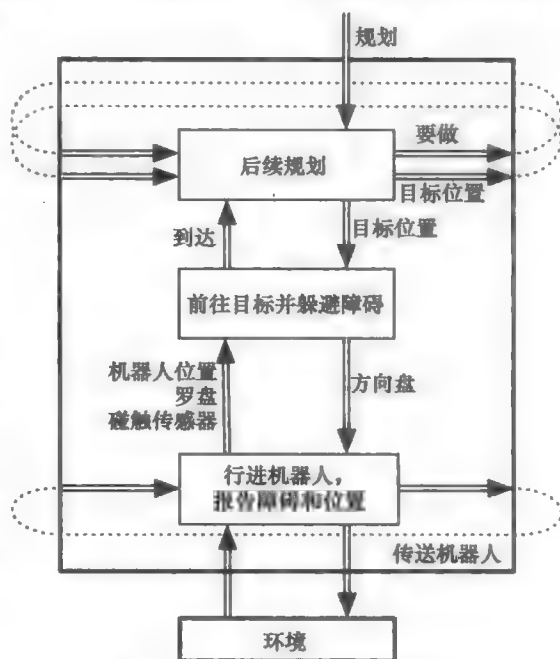


图 2-5 传送机器人的分层分解

【例 2-5】 前往目标并躲避障碍的中间层操纵机器人来躲避障碍物。此层的输入与输出在图 2-6 中给出。

机器人有一个通过触摸来发现障碍物的触觉传感器。由下层提供的一个比特数值可以判定触觉传感器是否碰触到障碍。底层也提供机器人的当前位置和方向。机器人所有能做的事情只是向左或向右旋转一个角度，或者直行。而此层的目的是让机器人向当前目标位置运行，并躲开路上的障碍，到达时发出到达报告。

控制器的此层不包含内部信念状态，因此并不存在信念状态转换函数。而用于控制机器人转向的命令函数则是其输入和是否已到达的函数。

当机器人的当前位置接近于上一个目标位置时说明其已到达。因此 *arrived* 状态会被指定一个值，它是机器人当前位置、前一个目标位置与一个阈值常数共同决定的函数：

$$arrived := distance(previous_target_pos, robot_pos) < threshold$$

其中， $:=$ 是指定，*distance* 是欧几里得距离，*threshold* 是一个合适的距离。

如果触觉传感器开则左转，否则朝向目标位置。这可以通过给 *steer* 指定合适的值来完成：

```

if whisker_sensor = on
  then steer := left
else if straight_ahead(robot_pos, robot_dir, current_target_pos)
  then steer := straight
else if left_of(robot_position, robot_dir, current_target_pos)
  then steer := left
else steer := right
end if

```

当机器人位于 *robot_pos*，面朝方向 *robot_dir*，且当前目标位置 *current_target_pos* 在机器人的前方位置，并满足一些阈值限制时（对于后面的例子，此阈值为正前方偏移 11° ），*straight_ahead(robot_pos, robot_dir, current_target_pos)* 为真。函数 *left_of* 用于检测目标是否在机器人左侧。

此层是纯定量化的。它以数字量化推理而非使用离散值。

【例 2-6】 最顶层，也叫后续规划层，会被给予一个规划，该规划为一个需要顺序访问的地点的名单。其中都是各种目标，这些目标都可被规划者开发出来，如在第 8 章中开发出的那些。当机器人到达前一目标时顶层也会被告知。它必须将当前目标坐标输出给中间层，并记住执行规划时其需要的。此层显示在图 2-7 中。

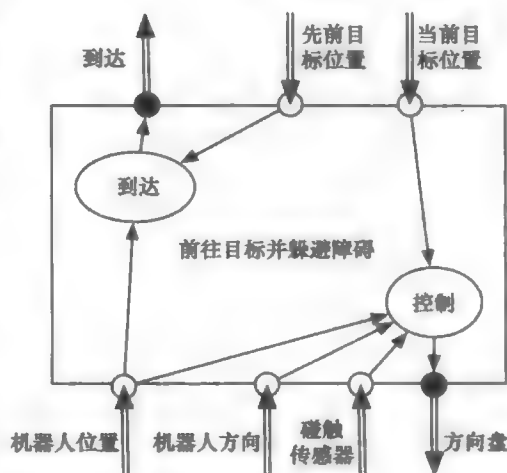


图 2-6 传送机器人的中间层

55

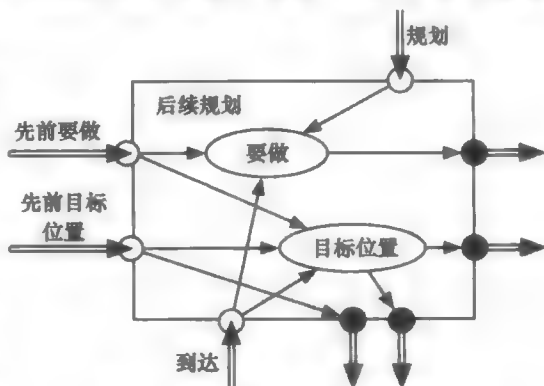


图 2-7 传送机器人控制器的顶层

56

该层会保留内部信念状态。它会记住当前目标位置和其他需要去访问的位置。其中 *to_do* 特征的数值是待访问的地点的一个列表。而 *target_pos* 特征保留着当前目标的位置。

一旦机器人到达它的前一目标位置, 下一个目标位置就会变成下一个需要访问地点的坐标。高层规划会以位置名字的形式给予机器人, 因此必须将其转换成中间层可以使用的坐标形式。下面的编码说明了当机器人到达前一目标位置时目标位置和 *to_do* 列表如何改变。

```
if arrived and not empty(to_do)
  then
    target_pos' := coordinates(head(to_do))
    to_do' := tail(to_do)
  end if
```

其中, *to_do'* 是 *to_do* 特征的下一个值, *target_pos'* 为下一个目标位置。在这里 *head(to_do)* 是 *to_do* 列表的第一个元素, *tail(to_do)* 是余下的 *to_do* 列表, 当 *to_do* 列表为空时, *empty(to_do)* 为真。

在这层中, 如果 *to_do* 列表变成空, 机器人就不会改变其目标坐标, 之后便会一直在原地绕圈。见习题 2.3。

此层可确定已命名地点的坐标。通过一个简单的有位置坐标的数据库就可以实现。如果地点不会移动且事先知道, 使用这样一个数据库是很明智的。然而, 如果目标可以移动, 那么下层必须能告诉上层目标当前位置。上层必须请求下层告知给定目标的坐标。见习题 2.8。

为了完成控制器, 还必须初始化信念状态变量, 且高级规划必须输入。通过使用规划的尾部来初始化 *to_do* 列表和使用第一个位置的坐标来初始化 *target_pos*, 这就可以实现。

图 2-8 给出了有一个障碍的规划 [*goto(o109)*, *goto(storage)*, *goto(o103)*] 的仿真。机器人开始时位于位置 (0, 5) 且面朝 90 度(北), 而在位置 (20, 20) 与 (35, -5) 之间有一个矩形障碍。

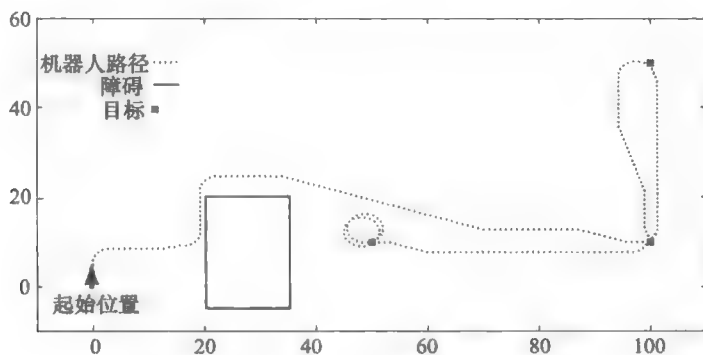


图 2-8 执行例 2-6 中规划的机器人仿真

Agent 对世界的建模

信念状态的定义非常一般, 且没有限制哪些是 Agent 应该记住的。一般来说 Agent 保留一些世界的模型是很有用处的, 尽管这些模型不完全、不精确。一个关于世界的模型是

某一时刻世界状态和/或世界的动态的一种表示形式。

对于 Agent 来说,可以基于其命令来保持和更新世界的信念。这种方法需要世界的状态模型和世界的动态模型。给定某个时刻的状态和动态,可以预测下个时刻的状态。这一过程也被称为**推算**。例如,机器人可以保留其对自己位置的估计并基于自己的动作来自我更新。若世界是动态的或者执行器有噪声(如半径不精确的轮子的滑动或者加速不连续),噪声会累积,因此估算的位置很快就会变得不准确以至于无用。然而如果模型在某些抽象层次上是准确的,则会成为该抽象层次上的合适模型。

57

另一种方法是使用**感知**来构建世界的相关部分的模型。感知就是用传感器信息来理解世界。举个例子,这可能会包括使用视觉来分辨世界的特征和使用这些特征来决定机器人、障碍物或者需要捡起的包袱的位置。感知一般都是模糊的和有噪声的。仅基于一张世界的图片很难建立一个三维世界模型。

一个更加有前景的方法是将 Agent 对世界状态的预测与感知信息结合起来。这可以采用多种形式:

- 如果向前预测的噪声与感知的噪声都有建模,那下一个信念状态就可以使用贝叶斯规则来估算。这也被认为是**过滤**。
- 如果使用更复杂的传感器,如视觉传感器,那就可以使用模型来预测何处可以发现视觉特征,接着便可用视觉来查找预测位置附近的这些特征。这会使得视觉任务变得更加简单,且相对于前面的仅依靠预测,使用视觉可以大幅减少位置误差。

如果可以采用如下方式得到最优动作,即通过首先发现最佳的世界模型,然后再使用此模型来决定最佳动作,那么控制器问题便是**可分割的**。不幸的是,大部分控制问题是不可分割的。这意味着 Agent 需要考虑多种模型来决定接下来做什么,且它可从世界获得什么信息取决于其对该信息的不同处理。通常来讲,并不存在与 Agent 行为完全独立的世界最佳模型。

58

2.4 嵌入式和仿真 Agent

Agent 控制器有许多可使用的方法:

- **嵌入式 Agent** 是一个可以在实际世界中运行的 Agent,其行为会在一个实际领域内执行,感知也来自此领域。
- **仿真 Agent** 是一个运行在模拟主体和环境中的 Agent,即一个可以接收命令并返回适当感知的程序,经常用于控制器实际实现之前进行纠错。
- **Agent 系统模型** 是一个包括控制器模型(这个不能确定是否为真实编程)、主体模型和环境模型,它可以回答 Agent 会有何种动作。这样一个模型可用于 Agent 创建前证明其性质,或者用于回答那些实际 Agent 很难或者无法回答的假说问题。

这里的每一种都有不同的适应目的。

- 嵌入模式适用于 Agent 必须实际使用的情况。
- 当有很多种设计选择需要实现,且构建一个实体又比较昂贵或者环境比较危险、不易见到时,仿真 Agent 更适用于测试和纠错控制器。它也允许我们在现实中难以实现的、非寻常情况组合下的环境中测试 Agent。

仿真情况如何主要取决于环境模型的好坏。模型总是会抽象世界的一些特征。合适的抽象对于仿真来说是很重要的,它能告诉我们 Agent 是否能在真实环境中工作。

- Agent 的模型、可能环境集的模型和一个特定的正确行为可以允许我们证明命题,

即关于 Agent 如何在这样的环境中工作。例如, 我们可能希望证明使用特定控制器的机器人总是可以到达目标确定距离范围内, 且一定不会迷路, 不会崩溃。当然, 所证实的是否正确取决于模型是否准确。

- 给定一个 Agent 和环境的模型, Agent 的某些方面可以先不指定, 之后可调整以生成所希望的或者最优的行为。这是优化与规划中的常用方法。
- 经过强化学习, 同现实世界交互时, Agent 可展现出更优的性能。

59

2.5 通过推理来行动

之前的部分假定 Agent 有着很多在各个时段保留下来的信念状态。对于一个智能 Agent, 信念状态可以非常复杂, 甚至可用占用单独一层。

学习和构建智能 Agent 的经验表明智能 Agent 需要某些信念状态的内部表示方法。知识指的是领域内的信息, 可用于解决该领域内的问题。知识指的是用于特定情境、领域的一般性知识。因此关于特定状态更常使用知识而非信念。**基于知识的系统**是可以使用领域内知识来动作或者解决问题的系统。

哲学家们将知识定义为正确的、经过证实的信念。人工智能研究者们则更倾向于融合两种概念。知识更倾向于已被证实的一般性信息。而信念则是可以被新的信息修正的信息。一般信念都会伴随着可信度和如何交互的模型。在人工智能系统里, 知识不必是真的, 且只在有用时才去证实。这点区分经常会变得模糊不清, 因为经常会有 Agent 的一个模型将某种信息看做真, 而另一个模型则认为这些信息需要继续修正。

图 2-9 是图 1-3 的精练表示, 针对的是基于知识的 Agent。知识库是离线建立的, 但被用于在线产生动作。Agent 的这种分解与 Agent 的分层观点是正交设计; 智能 Agent 既需要分层组织又需要知识库。

在线是指, 当一个 Agent 动作时, 它会使用自己的知识库、对世界的观察、目标和能力来共同选择去做什么及更新知识库。知识库是**长期记忆**, 这里存储着将来动作时会用到的知识。这些知识来自于先验知识以及从数据和先前经验所学的结合。**信念状态**则是 Agent 的**短期记忆**, 它保留着步长之间所需要的当前环境模型。一般性知识与专门知识之间并不总是存在明显的界限; 比如, 一个户外的传送机器人可以学习特定城市的一般性知识。从推理引擎到知识库是存在反馈的, 因为在环境中观察和行动可以提供更多的可供学习的数据。

离线, 指的是在 Agent 有所动作之前, 它能构建一个对其在线行动时有用的知识库。离线计算的任务是使得在线计算更加有效、高效。知识库是通过先验知识和先前经验得来的数据(既可是它自己的先前经验, 也可是别人给予的数据)来构建。**机器学习**领域中的研究者们已经考虑过有很多数据和仅一点先验知识的情况。而**专家系统**中研究了有很多先验知识和很少甚至没有可用于学习的数据的情况。然而, 对于几乎所有非平凡领域, Agent 必须使用所有可用的信息, 因此既需要丰富的先验知识又需要大量数据。

目标和能力是离线赋予、在线赋予还是都赋予, 主要取决于 Agent。举个例子, 一个

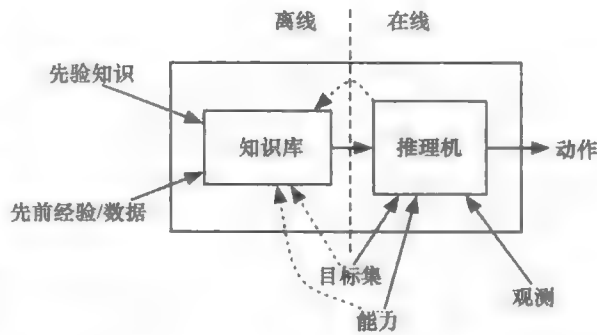


图 2-9 Agent 的离线和在线分解

60

传送 Agent 可能有保证实验室清洁并不损坏自己或者其他物品的一般性目标，但是它也可在运行中获取其他传送目标。如果知识库被调整为适合特定目标和能力，那么在线计算会变得更加高效。然而，当不能获知运行中的全部目标和能力时，这一般来说不大可能。

图 2-10 更为详细地显示了 Agent 与环境的交互接口。

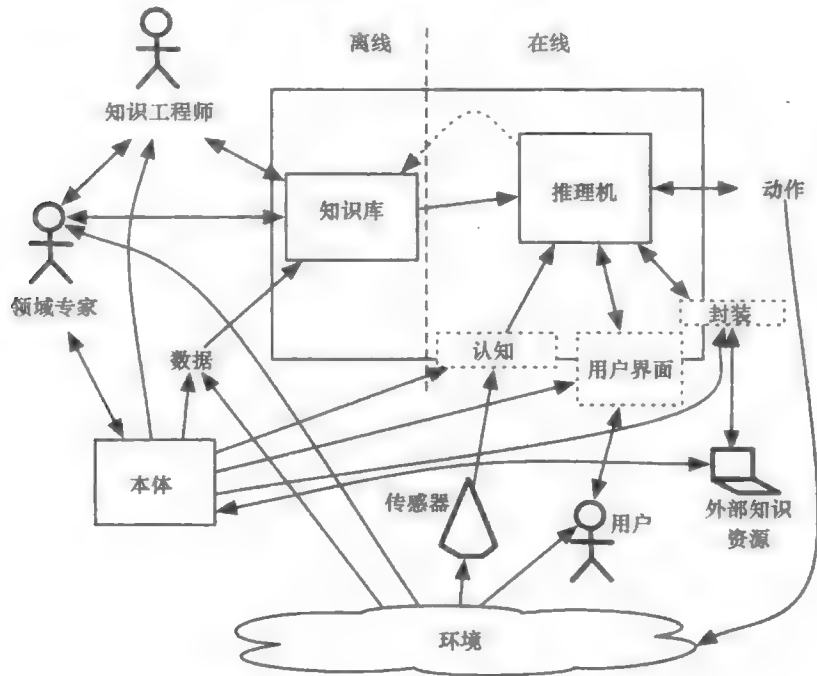


图 2-10 Agent 的内部，职能说明

2.5.1 设计时间与离线计算

在线计算需要的知识库可以在设计时便初始化，然后由 Agent 离线扩张。
本体是用于信息系统的符号含义的一种描述。它指定了建模什么及系统中使用的词汇。最简单情况，如果 Agent 正在使用完全观察的明确的基于状态的表述，那么本体就表示环境和状态间的映射方式。如果没有这个映射，Agent 也许知道其在状态 57，但是如果缺少本体，对于其他 Agent 或者人来说，这个信息毫无意义。在其他情况下，本体同样能定义特征或者个体和关系。它被用于将未加工的感知转换为对于 Agent 有意义的信息，或者用于从人类或者其他知识源处获得有意义的输入。

本体是由社区构建的，一般独立于特定的知识库和特别的应用。正是由于此共享特性才允许使用多种来源(视觉、人类、数据库)的数据进行有效的沟通和内部操作。个体和关系情况的本体在 13.3 节讨论。

本体逻辑上应早于数据和先验知识：我们需要一个本体来获取数据或者知识。没有本体的话，数据仅是比特串。没有本体，人们不知道应输入什么；正是本体给数据赋予了含义。一般来说本体会在系统开发时逐步完善。

本体会指定一个或多个抽象层。如果本体发生改变，那数据必须改变。例如，机器人应该有关于障碍物的本体(如任何实体对象都是应该避开的障碍物)。如果再被扩展到可区分的人类、椅子、桌子、咖啡杯或者其他类似东西，那么就需要更多的关于环境的不同

数据。

知识库是离线地通过结合专家知识和数据来构建。其通常在 Agent 了解将活动的环境前就已构建。在线计算的部分通常包括保存并调整知识库。

离线时, 涉及基于知识系统的有三个主要任务:

- **软件工程师**构建推理引擎和用户接口。他们通常并不知道知识库的内容。在使用他们所实现的系统方面, 他们并不需要很专业; 然而, 在编程语言, 如 Java、Lisp、Prolog 的使用方面, 而非他们自己所设计系统的知识表示语言方面, 他们必须是专家。
- **领域专家**是那些在某一领域有适当先验知识的人。他们了解相关领域, 但是通常来说他们对于一些可能会遇到的特殊情况一无所知。例如, 一个药学领域的专家知道疾病、症状及相应关联, 但是可能却不知道某个特定病人的症状或者疾病。一个传送机器人领域的专家也许知道必须认识的个体种类、电池仪表的意义与各种动作的耗费率。领域专家通常并不知道 Agent 会遭遇的环境的详细信息, 比如, 对于诊断助手来说的病人的详细信息, 或者机器人所在房间的详细信息。

领域专家通常并不清楚人工智能系统的内部工作过程。一般他们仅有关于知识的语义观点且并不关心推理引擎在使用何种算法。系统应以领域的形式与他们进行交互, 而非计算步骤的形式。例如, 若一个知识库采用显示答案产生轨迹的方式, 然后寄希望于领域专家来修正此知识库是不明智的。因此, 并不需要提供给领域专家修正工具来跟踪程序的执行。

- **知识工程师**在同领域专家协商的基础上设计、构建和修正知识库。他们了解系统的详细信息且通过领域专家了解领域内的详细信息。但是他们对于具体案例毫不了解。他们应该知道有用的推理技术和完整系统的工作流程。

同样一个人可以完成多个任务: 一个了解人工智能的领域专家可以作为知识工程师; 一个知识工程师可以作为编写系统的人。一个大的系统可以有許多不同的软件工程师、知识工程师和专家, 其中每一个人专门执行系统的一部分。这些人可能甚至不知道他们仅是系统的一部分, 他们发布的信息可被任何人使用。

离线时, Agent 可以结合专家知识和数据。在这个阶段, 系统可以被测试和修正。Agent 能够计算那些非特殊指定情况。例如, 它能编译部分知识库来使得推理更加高效。

2.5.2 在线计算

在线状态时, 具体事例的信息开始可用, 且 Agent 需要开始行动。这些信息包括领域的观察信息以及可行动作、偏好和目标的信息。Agent 可以从传感器、用户和其他信息源处(如网站)获得信息, 但是我们假定它不能从领域专家或者知识工程师处获得信息。

一般来说 Agent 的离线计算时间要远大于在线计算时间。然而, 在线计算时它能更好地利用特定目标和特定的观察。

例如, 一个医药诊断系统仅当在线状态时才有特定病人的详细信息。离线状态时, 它能获知疾病和症状相关性的信息, 并能对知识进行修正和完善。仅当它在线状态时, 才能处理特定病人的相关信息。

在线主要包括以下几个任务:

- **用户**是一个需要专业知识的人或者有个体案例信息的人。用户通常不是知识库相关领域的专家, 所以他们一般并不知道系统需要什么样的信息。因此不可能奢望

他们能自主地提供特定案例的信息。这时候就需要一个简单自然的界面,因为用户通常并不知道系统的内部结构。他们一般会根据系统给出的建议做一个相对全面的决定,因此需要对所给的建议给出一个合适的解释。

- **传感器**会提供环境的信息。例如,温度计是一个可以提供其所处位置当前温度的传感器。传感器也可以更加复杂,如视觉传感器。在最底层,视觉传感器简单地提供每秒 30 帧的大小为 720×480 像素的矩阵。而在更高层,视觉系统也许能够回答一些特定问题,如特定特征的位置、某种物品是否存在或者视野中是否存在某些特定的个体。一个麦克风的数组可被用在底层抽象级上,用来提供详细的震动信息。它同样可以被用做高层传感器的组件来检测爆炸,并预测爆炸的类型和位置。

传感器主要分为两种。一种是**被动传感器**,总是连续的发送信息给 Agent。被动传感器包括温度计、相机和麦克风。设计者可以选择传感器的放置位置和方向,但是他们仅能提供 Agent 信息。而另一种**主动传感器**则恰恰相反,它可被控制或者可向其咨询信息。主动传感器的例子有很多,如可以回答关于病人的特定信息的医用探针,或者智能辅导系统中的一份给学生的测试。经常会出现一个传感器在底层抽象级上是被动传感器,而在高层抽象级上被看为主动传感器。例如,相机可被询问房间内是否存在特定人物。为了完成此任务,它必须找出房间内的面孔并找出每个人的不同特征。

64

- **外部知识源**,如一个网站或者数据库,一般能够回答问题并且在受限领域提供答案。Agent 可以向一个天气网站询问某一地点的温度,也能向一个航空网站询问指定航班的到达时间。知识源有不同的协议和有效的权衡机制。Agent 与外部资源间的接口被称为**封装**,封装可以互相转换 Agent 使用的表达信息与外部知识源准备去处理的查询信息。一般来说封装的设计意义在于一个 Agent 可以向多个知识源询问同一事情。例如,一个 Agent 也许希望知道飞机的到达时间,可是不同的航线或者飞机场需要不同的协议接口进入以获得此信息。当一个网站或者数据库通用一个共同的本体时,他们就能互相使用信息了,因为相同的符号代表相同的意义。相同符号代表相同事情这也叫做**语义互通**。当他们使用不同的本体时,那么在本体间必须存在映射关系以便能够实现互通。

而且,任务这样便分开了,尽管完成这些任务的人群之间可能会重叠。例如,领域专家可以作为一个用户来测试和修正系统,每个任务对他们需要的工具来说都有不同的要求。那些用于向用户解释系统如何得出结果的工具,也可以是领域专家用来修正知识的工具。

2.6 本章小结

- 一个 Agent 系统由一个 Agent 和一个环境组成。
- Agent 需要传感器和执行器来与环境进行交互。
- 一个 Agent 由主体和相关控制器组成。
- Agent 是适应于时间,且必须根据其以往与环境的交互历史来决定做什么。
- 一个 Agent 不会直接访问其历史,而是它所记住的内容(信念状态)和它刚刚观察到的信息。在每一个时刻,Agent 对于去做什么和记下什么主要取决于其信念状态和当前的观察。
- 复杂的 Agent 以可交互的分层结构形式构建。

65

- 智能 Agent 是需要知识的, 而知识在设计阶段、离线阶段、在线阶段均可获得。

2.7 参考文献及进一步阅读

Agent 系统的模型是基于 Zhang、Mackworth[1995]的关于约束网络的工作和 Rosenschein、Kaelbling[1995]的工作, 分层控制则是基于 Albus[1981]的工作和 Brooks[1986]等人的关于包含结构的工作。Abelson、DiSessa[1981]的 *Turtle Geometry* 从建模简单反应式 Agent 角度介绍了相关数学理论。Luenberger[1979]则是一篇值得一读的关于 Agent 与环境交互的经典理论的简介。Simon[1996]则介绍了分层控制的重要性。

更多的 Agent 控制细节可以参看 Dean 和 Wellman[1991]、Latombe[1991]和 Agre[1995]。

构建智能 Agent 的方法论在 Haugeland[1985]、Brooks[1991]、Kirsh[1991b]和 Mackworth[1993]中多有介绍。

定量推理在 Forbus[1996]和 Kuipers[2001]中有讲述。Weld 和 de Kleer[1990]中有很多关于定性推理的精华, 同样, Weld[1992]也是关于此问题的。至于近期的综述可查看 Price、Travé-Massuyàs、Milne、Ironi、Forbus、Bredeweg、Lee、Struss、Snooke、Lucas、Cavazza 和 Coghill[2006]。

2.8 习题

2.1 2.3 节提到一个论点, 即不可能脱离 Agent 的任务和目的而独立地构建一个世界表示。此习题能使你评估此论点。

选择一个特定的世界, 如当前你的桌面的一部分。

i) 让某个人列出在此世界上存在的所有东西(或者自己完成)。

ii) 再考虑 20 种没有列出的, 且尽可能相互完全不同。例如, 桌子最右边的圆珠笔的笔头上的小球、订书机的弹簧, 或是桌上某本书 66 页的第三个单词。

iii) 找出一个无法用自然语言描述的东西。

iv) 选定一个任务, 如整理桌面, 并写出所有与此任务相关的所有物体的描述。

基于此习题, 讨论如下几个状况:

(a) 世界上存在什么是由观察者观察来决定的。

(b) 我们需要某种方法来指定到每一个个体, 而非期望每个个体都有独立的名字。

(c) 存在哪些个体是由任务同样也是世界的属性决定的。

(d) 为了描述领域中的某个个体, 你需要有一个极大的词典和一个有效的方法来组合它们用于描述个体, 而且这点必须与任何特定的领域无关。

2.2 解释为何例 2-5 中的中间层必须有前一目标位置和当前目标位置作为输入。假定只有其中一个作为输入, 哪一个是必需的? 这将会导致什么问题?

2.3 例 2-6 中的目标位置的定义意味着当规划结束时, 机器人将会将其最后一个目标位置作为其目标位置, 然后不断绕圈。改变定义使得机器人可以返回原点, 然后围绕原点绕圈。

2.4 如例 2-5 中的方法躲避障碍物很容易便会陷入困局。

(a) 设置一个障碍物和一个目标以便于机器人使用例 2-5 中的控制器, 会发现其不能到达(会崩溃或者绕圈)。

(b) 即使没有障碍物, 机器人也可能无法到达目的地。例如, 如果他在目标位置附近, 它就会不断绕圈而无法到达目标地点。设计一个可以发现此问题并能使其到达目标点的控制器。

2.5 考虑图 2-11 中的“机器人困境”。

(a) 解释一下为何它会使一个机器人难以到达目标点 g。你必须解释机器人现在会有何种行为, 且为何难以设计一个更为复杂的

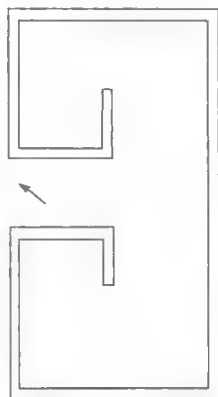


图 2-11 一个机器人困境

66

机器人(如,一个使用“右手规则”的机器人会沿着墙走:当其碰到障碍物时其会左转并沿着墙走,而墙面始终在其右侧)来工作。

67

- (b) 直观上我们认为如果想要脱离这个迷宫,需要当一个机器人撞到墙后,它会一直绕着墙走直到右转次数与左转次数相同。告诉我们如何实现此方法,解释信念状态、信念状态转换函数和命令函数。

2.6 当用户选择并移动当前目标位置时,此章中的机器人会继续行至此目标的原坐标处,而不会再去其当前位置。试着改变此控制器使其可以向目标的当前位置移动。

2.7 现在的控制器会顺序访问 *todo* 列表中的地点。

- (a) 改控制器为投机型,当它需要选择下一个位置时,它会选择距离当前位置最近的目的地。当然其仍需访问所有的目的地。

- (b) 给出一个环境实例,在此环境中新控制器完成访问任务花费的时间少于使用原控制器花费的时间。

- (c) 给出一个环境实例,在此环境中使用原控制器花费的时间要少于使用修改后的控制器所花费的时间。

- (d) 改变控制器,使得 Agent 在行进的每一步都朝向距其当前位置最近的目的地。

- (e) 使用(d)中描绘的控制器会不会出现陷入死循环以致无法到达目的地,而使用原控制器却可以正常工作的环境中? 给出一个其会陷入原地转圈的例子并解释为何其无法找出一个解决方法,或者给出其为何不会陷入转圈的原因。

2.8 改变控制器使得机器人可以感知环境并获知某位置的坐标。这里假定主体可以提供已命名位置的坐标。

2.9 假设你有个新工作,必须为一个机器人构建一个控制器。你告诉老板你只需要实现命令函数和状态转换函数。他们会对此产生质疑。为何是这些函数? 只需要这些函数? 向其解释为何一个控制器仅需要一个命令函数和一个状态转换函数。请用适当的言语,要简洁。

68

第二部分

Artificial Intelligence: Foundations of Computational Agents

表达和推理

状态和搜索

你看到过海岸上的螃蟹在寻找大西洋的过程中一直向后爬行，最终消失吗？人们也是同样的思考方式。

——H. L. Mencken (1880—1956)

前面一章讨论了 Agent 是如何理解和动作的，但是没有讲到它们的目标是如何影响动作的。一个 Agent 可以根据既定的目标集有规划的去动作，但是如果不能适应变化的目标，这样的 Agent 就是非智能的。或者，Agent 能够根据自身能力和既定目标去推理，从而决定应该做什么。这一章介绍将 Agent 决定做什么的问题描述为在一个图中搜索发现一条路径的问题，它还给出了计算机解决此类问题的很多方法。就像 Mencken 在上面说的，尽管不都能达成既定目标，人们还是想利用搜索的方法去解决问题。

3.1 用搜索进行问题求解

在 Agent 推理要做什么的最简单的例子中，Agent 有一个基于状态的世界模型，没有不确定性，并具有要完成的目标。这个模型或者是扁平的(不分层的)表示，或者是单层结构。Agent 能够通过对世界状态空间表示的搜索找到从当前状态到目标状态的一条路径，从而决定如何完成其目标。它能在其行动之前找到实现其目标的一个动作序列。

这个问题可以抽象为一个数学问题，在一个有向图中找到从起始节点到目标节点的一条路径。其他很多问题都可以映射成这类抽象，所以考虑这个层次上的抽象很有意义。本章大部分内容研究了寻找这样路径的各种算法。

搜索的含义是指 Agent 内在的计算。它不同于必须要有动作的现实世界的搜索，例如，寻找钥匙、举起垫子等。它也不同于网络上的信息检索。这一章中所研究的搜索是在内部描述中找到一条到达目标的路径。

搜索的思路是明确的：Agent 针对一个问题构造出一个可能的局部解的集合，可以检查它们是否是最终解或者可以导致最终解。搜索就是反复地选择局部解，直到找到一条通往目标的路径才停止，否则，就会在所有可能的方向增加一条弧来扩展局部解。

搜索是大部分人工智能领域的基础，当 Agent 遇到一个问题时，它只给定了可以识别解的描述，而不知道解决这个问题的具体算法，所以它必须搜索问题的解。在 NP 完全问题中，我们有有效的方法来识别答案，却没有有效的方法去找到答案。这说明，在许多情况下，搜索是解决问题的必要部分。

人们经常能够通过直觉一步就找到很难的问题的解。但是，人们不倾向于解决普遍问题，相反，他们会处理具体的问题，与可能的搜索空间相比，他们熟悉这些问题。有些问题中不存在结构，或者其中的结构与物理世界不相关，这样的问题人们很难解决。在公钥加密编码中，搜索空间清晰明了，解的验证方法也已给出，尽管如此，人类仍然无法破解，计算机也不能在现实的时间框架中破解，这就例证了搜索的困难性。

搜索的困难性及人类能够有效解决一些搜索问题的事实，表明计算机 Agent 应当从一些

特例中拓展新知识来指引它们找到解。超出搜索空间以外的知识叫做启发性(式)知识(heuristic knowledge)。本章以评估从一个节点到一个目标的花费的形式讲述一种启发式知识。

3.2 状态空间

智能行为的一种一般形式就是状态空间(state space)。一个状态(state)包含着所有必备的信息,以用来预测动作结果并确定是否是目标状态。状态空间搜索假设如下:

- Agent 具有状态空间的完全知识,并可观察自己所处的状态(即完全可见)。
- Agent 具有一个动作集,这些动作有已知的确定效果。
- 一些状态是目标状态,Agent 想要到达其中一个目标状态,并可识别目标状态。
- 解(solution)就是一个动作序列,使得 Agent 从当前状态到达目标状态。

【例 3-1】图 3-1 显示了机器人投递域及寻找从一个位置到达另一个位置的一条路径的任务。这可以抽象为一个状态空间的搜索问题模型,其中状态就是那些位置。假设 Agent 能够利用低级控制器去执行从一个位置到相邻位置的高级行为,那么在这种层次的抽象中,行为就是指相邻位置的明确的移动。

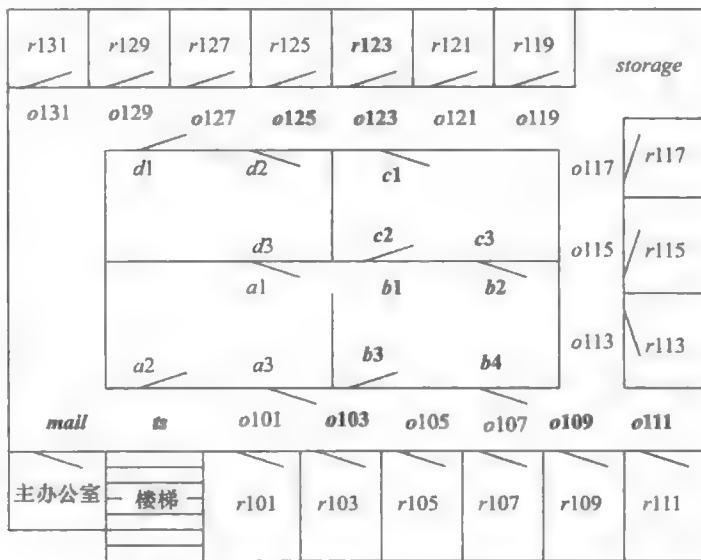


图 3-1 标明感兴趣位置的投递机器人域

例如,机器人在房间 r103 外,即图中 o103 位置处,目标是到达房间 r123。解就是使机器人到达 r123 的动作序列。

【例 3-2】在更复杂的情况下,投递机器人可能有多个包裹要投送到不同的位置。在这种情况下,状态包含:机器人的位置、机器人正在投送的包裹以及其他包裹的位置。机器人可能做出的动作包括:移动、捡起与自己在同一位置的包裹,或者随意将包裹放下。目标状态则可能是将一些特定包裹送达预期的位置。因为我们不考虑机器人或者其他一些包裹的位置,所以可能会出现很多目标状态。

注意上述描述忽略了很多细节,例如,机器人怎样携带包裹(这会影响到它是否能携带其他包裹),机器人所剩的电量,包裹是否易碎或者损毁,以及地板的颜色。我们不把它们作为状态空间的一部分,而假定这些细节与目前所研究的问题无关。

【例 3-3】在一个指导系统中,一个状态可能包含着学生知道的主题集合。动作就是

教授一门特定课程，教学行为结果可能是学生弄懂课程的主题，因为学生知道这些主题是他们掌握课程的先决条件。目标就是学生能够懂得一些特定的主题集合。

如果教学的效果也取决于学生的能力，那么这个细节信息也必须作为状态空间的一部分。如果学生正在做什么并不影响行为的结果，也不影响目标是否能达到，那么我们就未必为它建模。

一个状态空间问题(state-space problem)由以下部分组成：

- 一个状态集；
- 一个特定状态集合，称为初始状态(start state)；
- 每一个状态中 Agent 可执行的一个动作集；
- 一个动作函数(action function)，给定一个状态和一个动作，返回一个新的状态；
- 一个目标状态集，通常被定义为布尔函数， $goal(s)$ ，当 s 为目标状态时，这个函数返回值为真。
- 用于检验一个可接受解的质量的标准。例如，只要能让 Agent 到达目标状态的任意一个行为序列都是可接受的解，或者动作是有花费的，Agent 可能被要求找到最小总花费的解，这样的解叫做最优(optimal)解。或者说，比最优解花费多 10% 以内的解都是令人满意的。

这种框架在接下来的章节中会进一步扩展讲解，例如以下情况，Agent 可以发掘状态的内在特征、状态不能完全可观察(例如，机器人不知道包裹在哪，或者老师不知道学生的能力)，或者动作是随机的(例如，机器人有可能跑过了目标位置，或者学生可能不学习被教的主题)，或者不仅仅是到达目标状态，还涉及奖惩时存在着复杂的偏好，等等。

3.3 图搜索

在这一章中，我们将一般搜索机制抽象表示为在有向图中的路径搜索。要解决一个问题，首先定义潜在的搜索空间，然后将搜索算法应用于其中。许多问题求解的任务都可以转化为在图中寻找路径的问题。图搜索提供了抽象的一个合适层次，用这种抽象，可以研究独立于特定领域的简单问题求解。

一个(有向)图包含一个节点集及一个节点间的有向弧集。其思路是找到一条沿着这些弧从起始节点到目标节点的路径。

因为将问题表示为图不止一种方式，所以抽象很重要。这一章中的例子都是有关状态空间搜索，节点代表状态，弧代表动作，以后的章节中会讨论用图表示问题的不同搜索方式。

形式化图搜索

一个有向图(graph)包括：

- 一个节点(node)集合 N ；
- 一个节点的有序对集合 A ，叫做弧(arc)。

在这个定义中，节点可以是任何东西。定义的实质是约束弧是节点的有序对。可以有无限多的节点和弧。我们并没有假设图完全明确地表示出来，我们仅需要有一个进程根据需要产生节点和弧。

弧 $\langle n_1, n_2 \rangle$ 是一条从 n_1 的出弧(outgoing arc)和一条至 n_2 的入弧(incoming arc)。

如果有一条从节点 n_1 到节点 n_2 的弧线, 也就是 $\langle n_1, n_2 \rangle \in A$, 节点 n_2 就是节点 n_1 的邻居(neighbor)。注意到两点是邻居并不意味着对称, 因为 n_2 是 n_1 的邻居并不意味着 n_1 就必须是 n_2 的邻居。例如, 弧可能被标记(label)为使 Agent 从一个状态到达另一个状态的动作。

一条从节点 s 到节点 g 的路径(path)是一个节点序列 $\langle n_0, n_1, \dots, n_k \rangle$, 节点 $s = n_0$, $g = n_k$, $\langle n_{i-1}, n_i \rangle \in A$, 即对于每一个 i 都有从 n_{i-1} 到 n_i 的弧。有时将路径看做弧的序列很有用, $\langle n_0, n_1 \rangle, \langle n_1, n_2 \rangle, \dots, \langle n_{k-1}, n_k \rangle$, 或者是这些弧的标记序列。

一个环(cycle)是一条起点和终点相同的非空路径, 即一个环是一条路径 $\langle n_0, n_1, n_2, \dots, n_k \rangle$, $n_0 = n_k$ 且 $k \neq 0$ 。一个有向图如果没有任何环就称作有向无环图(directed acyclic graph, DAG)。这可能应该是一个无环的有向图, 因为它是有向图但是恰好无环, 而不是无环图正好有向, 但是有向无环图(DAG)比无环有向图(acyclic directed graph, ADG)听起来更好一些。

树(tree)就是一个有向无环图, 其中有一个节点没有入弧, 其他每一个节点都正好有一条入弧。没有入弧的节点叫做树的根(root)节点, 没有出弧的节点叫叶(leaves)节点。

要将问题编码为图, 一个节点集叫做起始节点(start node), 另一个节点集叫做目标节点(goal node)。解(solution)就是从起始节点到目标节点的一条路径。

75

有时会有一个与弧相关的花费(cost), 是一个正数, 我们将弧 $\langle n_i, n_j \rangle$ 的花费记作 $cost(\langle n_i, n_j \rangle)$, 弧的花费会引起整个路径的花费。

给定一条路径 $p = \langle n_0, n_1, n_2, \dots, n_k \rangle$, 路径 p 的花费就是路径上弧的花费之和:

$$cost(p) = cost(\langle n_0, n_1 \rangle) + cost(\langle n_1, n_2 \rangle) + \dots + cost(\langle n_{k-1}, n_k \rangle)$$

最优解(optimal solution)是最低花费的解之一。也就是说, 如果存在一条从起始节点到目标节点的路径 p , 而不存在其他从起始节点到目标节点的路径 p' 使得 $cost(p') < cost(p)$, 那么这条路径就是最优路径。

【例 3-4】 再来看图 3-1 中描述的投递机器人寻找从位置 o103 到位置 r123 的路径问题。在这个图中, 感兴趣位置已经标注。为了简化问题, 我们只考虑加粗字体指示的位置并且一开始就限定机器人移动的方向。图 3-2 显示了结果图, 在这个图中节点代表位置,

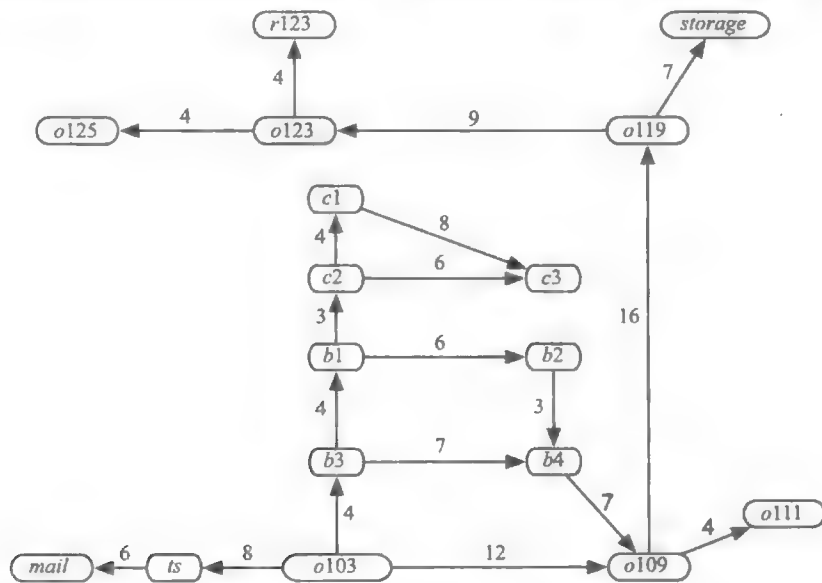


图 3-2 带有弧花费的投递机器人域的图

76

弧则表示机器人在位置间的可能做出的每一步移动，且每条弧线上标示出了从一个位置到下一个位置的花费。

在这个图中，节点集合 $N = \{mail, ts, o103, b3, o109, \dots\}$ ，弧集合 $A = \{\langle ts, mail \rangle, \langle o103, ts \rangle, \langle o103, b3 \rangle, \langle o103, o109 \rangle, \dots\}$ 。节点 $o125$ 没有相邻节点，节点 ts 有一个相邻节点，即 $mail$ 。节点 $o103$ 有三个相邻节点，即 ts 、 $b3$ 和 $o109$ 。

这样从 $o103$ 到 $r123$ 就有三条路径：

$\langle o103, o109, o119, o123, r123 \rangle$

$\langle o103, b3, b4, o109, o119, o123, r123 \rangle$

$\langle o103, b3, b1, b2, b4, o109, o119, o123, r123 \rangle$

如果 $o103$ 是起始节点， $r123$ 是目标节点，那么这三条路径都是图搜索问题的解。 ◀

很多问题中搜索图没有明确给出，要根据需要动态架构。所有搜索算法都要求遵循的原则是：从一个节点生成它的邻居(子节点)，然后确定它是否是目标节点。

一个节点的前向分支系数(forward branching factor)是指离开该节点的弧(出弧)的数目，而它的后向分支系数(backward branching factor)则是指进入该节点的弧(入弧)的数目。这些系数提供了图的复杂度的测量标准。当我们讨论搜索算法的时间和空间复杂度时，假定分支系数最高值(自上而下)由一个常数来界定。

【例 3-5】 在图 3-2 中，节点 $o103$ 的前向分支系数是 3，即从 $o103$ 有三条出弧。节点 $o103$ 的后向分支系数是 0，即无入弧指向节点 $o103$ 。 $mail$ 前向分支系数为 0，后向分支系数为 1。节点 $b3$ 前向分支系数为 2，后向分支系数为 1。

分支系数很重要，因为它是决定图大小的关键因素。如果每一个节点前向分支系数是 b ，且是树型图，那么该图中有 b^n 个节点，当前节点离任意节点的弧的段数是 n 。 ◀

3.4 一个通用搜索算法

这一节描述在图中寻找解路径的一个通用算法。这个算法独立于任何特定的搜索策略和特定的图。

通用搜索算法背后的直观理念就是给定一个图、一个起始节点集和一个目标节点集，循序渐进地从起始节点探索路径。这要通过维持已经探索过的从起始节点开始的路径的边界(或外围)来实现。边界包含能构成从起始节点到目标节点的初始部分的所有路径。(见图 3-3，边界就是指通向灰色节点的那组路径。)最初，边界包括从起始节点开始的无出弧的那些平凡的(不重要的)路径。随着搜索的深入，边界扩展到未扩展节点，最终到达目标节点。为了扩大边界，搜索者继续挑选，从边界去除路径，从最新节点出弧扩展新的路径并将这些新路径加入边界。搜索策略就界定了在每

77

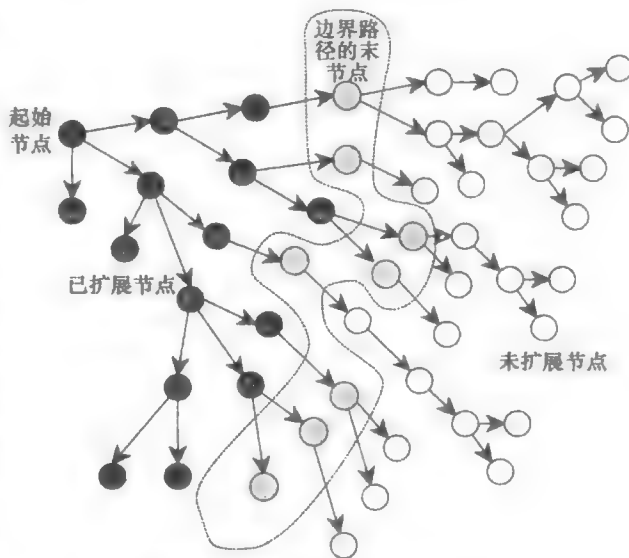


图 3-3 通过图搜索进行问题求解

一步中需要选择哪些边界要素。

图 3-4 中给出了通用搜索算法。最初，边界是一组从起始节点开始的空路径。每一步中，算法通过从边界去除路径 $\langle s_0, \dots, s_k \rangle$ 而继续推进边界。如果 $goal(s_k)$ 是真(即 s_k 是目标节点)，它就找到了解并返回已找到的路径，即 $\langle s_0, \dots, s_k \rangle$ 。否则就通过增加一条出弧来寻找 s_k 的邻居，从而扩展路径。因为对于 s_k 的每一个邻居 s ，路径 $\langle s_0, \dots, s_k, s \rangle$ 都会加入到边界中。这一步叫做扩展(expanding)节点 s_k 。

这个算法有一些要注意的特征：

- 13 行中路径的选择是不确定性的。路径的选择会影响效率，5.2.2 节中“不确定性选择”处有更多关于路径筛选使用的详细描述。特定的搜索策略决定选择哪一条路径。
- 15 行中的 return 应视为临时返回，通过继续进行的 16 行指令可得出另一条到达目标的搜索路径。
- 如果该过程返回 \perp ，则不存在解(或者即使再次试验进行验证也无其余解)。
- 这个算法只能用来测试从边界

```

1: procedure Search( $G, S, goal$ )
2:   Inputs
3:      $G$ : 具有  $N$  个节点和  $A$  条边的图
4:      $S$ : 开始节点集
5:      $goal$ : 布尔状态函数
6:   Output
7:     从  $S$  的一个成员到  $goal$  值为真的节点的一条路径，
8:     或者如果无解路径则  $\perp$ 
9:   Local
10:    边界: 路径集合
11:     $Frontier \leftarrow \{ \langle s \rangle : s \in S \}$ 
12:    While  $Frontier \neq \{\}$  do
13:      select and remove  $\langle s_0, \dots, s_k \rangle$  from  $Frontier$ 
14:      If  $goal(s_k)$  then
15:        return  $\langle s_0, \dots, s_k \rangle$ 
16:       $Frontier \leftarrow Frontier \cup \{ \langle s_0, \dots, s_k, s \rangle : \langle s_k, s \rangle \in A \}$ 
17:    return  $\perp$ 

```

78

图 3-4 通用图搜索算法

挑选的路径是否能终止在目标节点，而不是测试任意加入到边界的节点。对此有两点主要原因。有时边界上的节点到目标节点的弧花费很大，搜索就不应当通过这条弧返回路径，因为可能存在更低花费的路径。当要求最低花费路径的时候，这就显得尤为重要了。第二点原因是确定一个节点是目标节点的代价很大。

如果选择的路径不在目标节点处终止，并且终止节点没有邻居，那么扩展路径意味着去除路径。这种结果是合理的，因为这条路径不能看做是从起始节点到目标节点的路径的一部分。

3.5 无信息搜索策略

一个问题给定了图和目标，但并没有给出从边界中选择哪条路径。这是搜索策略的任务。一个搜索策略指定如何从边界中选择路径。通过改变在边界中选择路径的实现方法从而可以得出不同的搜索策略。

79

这部分介绍了三种不考虑目标节点位置的所谓无信息搜索策略(uninformed search strategy)。直觉上，这些算法不关心它们要到哪里去，直到找到目标节点并报告成功。

3.5.1 深度优先搜索

第一个策略是深度优先搜索(depth-first search)策略。在深度优先搜索中，边界的动作像一个先进后出的堆栈(stack)。这些元素被逐次地压入栈中。在边界中被选择并撤销的节点肯定是最后一个人栈的元素。

【例 3-6】 考虑图 3-5 中的树形图，假设初始节点是树的根（在顶部的节点），节点从左到右排序，所以最左边的邻居最后被压入栈中。在深度优先搜索中，节点扩展的方向并不依赖于目标节点的位置。图 3-5 中，我们将首批扩展的 16 个节点用数字标记出来了。阴影节点表示这 16 步检索完成后，边界路径上的最终节点。

注意最开始的 6 个节点是如何在单一的路径上扩展的。第 6 个节点没有邻居，因此，下一个要扩展的节点是距离它最近的且有尚未扩展的子节点的父节点的子节点。

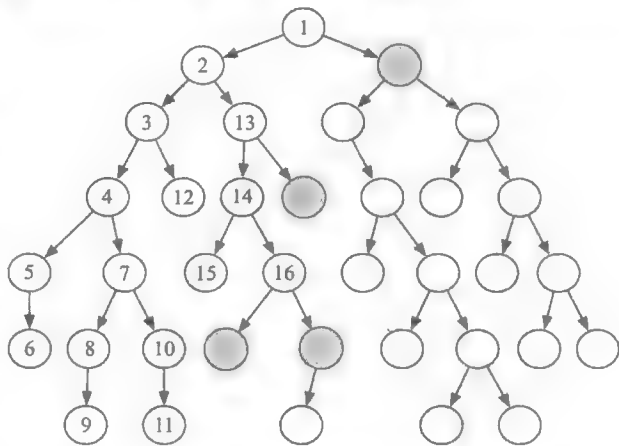


图 3-5 深度优先搜索中节点的扩展顺序

利用堆栈来实现边界会导致使用

深度优先的方式得到路径——在查找一条路径到头之后再尝试另一个分支路径。这种方法叫做回溯(backtracking)：在每一个节点，算法会先选择每个节点上第一条可选择的路径搜索下去，当执行完第一个选择的整个路径它会回溯到下一个可选的路径。一些路径可能是无限的，比如当图中有环或者有无限多节点时，这样深度优先搜索也许永远不会停止。

这种算法没有明确规定将邻居压入堆栈的顺序，并用堆栈代表了边界。算法的效率就跟这个顺序相关。

【例 3-7】 考虑图 3-2 中的从房间 o103 出发的深度优先搜索，唯一的目標节点是 r123。在这个例子里，边界表示为一个路径列表，栈顶作为列表的开始部分。

最初，边界包含平凡路径⟨o103⟩。

下一步，边界包含了后面的路径：[⟨o103, ts⟩, ⟨o103, b3⟩, ⟨o103, o109⟩]。

接下来，会选择路径⟨o103, ts⟩，因为它位于栈顶。这条路径就从边界中删除并用一段弧扩展它后再替换它，这样边界就变为：[⟨o103, ts, mail⟩, ⟨o103, b3⟩, ⟨o103, o109⟩]。

接下来，第一条路径⟨o103, ts, mail⟩也就被删除并被用一段新弧扩展后的一个路径集所替代，但由于 mail 节点没有邻居，所以该集合是空的。因此边界变为：[⟨o103, b3⟩, ⟨o103, o109⟩]。

在这个阶段，路径⟨o103, b3⟩在堆栈的顶部。注意一个事实，当深度搜索已经遍历了所有的从 ts 出发的路径(这里只有一条)，它就要回溯到堆栈的下一个元素。接下来，选择⟨o103, b3⟩，并用新弧的扩展路径替换它，边界变为：[⟨o103, b3, b1⟩, ⟨o103, b3, b4⟩, ⟨o103, o109⟩]。

然后从边界中选择⟨o103, b3, b1⟩并扩展，边界变为：[⟨o103, b3, b1, c2⟩, ⟨o103, b3, b1, b2⟩, ⟨o103, b3, b4⟩, ⟨o103, o109⟩]。

现在第一条路径就从边界中选出来了，并通过新弧扩展出来，产生边界路径：[⟨o103, b3, b1, c2, c3⟩, ⟨o103, b3, b1, c2, c1⟩, ⟨o103, b3, b1, b2⟩, ⟨o103, b3, b4⟩, ⟨o103, o109⟩]。

注意节点 c3 没有邻居，因此搜索回溯到最近的尚未造访的节点，即 c1。

假定⟨n₀, ..., n_k⟩是在边界中被选定的路径，边界的每一个其他元素都表示为

$\langle n_0, \dots, n_i, m \rangle$, 其中 $i < n$, 节点 m 是节点 n_i 的邻居。它沿着多个出弧确定的所选路径, 然后恰好有一个额外的点。

为了更好地理解深度优先的复杂度(参见下面“各种算法性能对比”的内容), 考虑使用家族树来类比, 一个节点的邻居对应的是树中的子节点。树的根是起始节点, 树的分支对应的是从起始节点开始的路径。考虑边界顶端的路径的最终节点, 边界的其他元素是这个节点父节点的其他子节点, 对应就是“叔叔”、“伯伯”, 等等。如果分支系数是 b , 列表的第一个元素的长度是 n , 那么最多有 $n \times (b-1)$ 个边界的其他元素。这些元素对应着每一个节点都有 $b-1$ 个可选路径。因此, 对于深度优先搜索, 使用的空间开销与从起始节点到某一个节点的路径长度呈线性关系。

各种算法性能对比

各种算法(包含搜索算法)可以按以下几项进行比较:

- 时间开销
- 空间开销
- 结果的质量以及准确性

时间开销、空间开销以及结果的准确性都是算法输入的函数。计算机科学家们所探讨的**渐进复杂度**(asymptotic complexity), 规定了时空开销是如何随着算法的输入量的增加而增长的。算法对于输入大小 n 有时间(或空间)的复杂度函数 $O(f(n))$, 读作“大 $O(f(n))$ ”, 这里 $f(n)$ 是关于 n 的函数, 如果存在常数 n_0 和 k , 对于所有的 $n > n_0$, 都有算法的时间或空间开销少于 $k \times f(n)$ 。这个函数的最常见形式是指数函数, 如 2^n 、 3^n 、 1.015^n ; 多项式函数, 如 n^4 、 n^2 、 n 、 $n^{1.2}$; 或对数函数, 如 $\log n$ 。一般来说, 指数函数算法比多项式函数算法复杂度增长更快, 相应的, 多项式函数算法比对数函数增长的快。

当输入大小为 n 时, 如果存在常数 n_0 和 k , 对于所有的 $n > n_0$, 都有算法的时间或空间开销大于 $k \times f(n)$, 就称算法有时间或空间复杂度 $\Omega(f(n))$ 。一个算法如果有复杂度 $O(n)$ 和 $\Omega(n)$, 那么它就有时间或空间的复杂度 $\Theta(f(n))$ 。通常情况下, 不能说一个算法的复杂度是 $\Theta(f(n))$, 因为大多数算法输入不同, 时间开销也不同。因此, 当比较算法时, 必须确定问题分类。

当通过时间、空间或准确性某个方面衡量出算法 A 比算法 B 更优, 则意味着:

- A 最坏的情况比 B 的最坏的情况好;
- 在实践中, A 能更好运行。或者在考虑典型性问题的平均情况时, A 运行的平均情况比 B 好;
- 由你描述哪类问题使用算法 A 比 B 好, 所以算法的优劣取决于要解决的问题;
- 对于任何的问题, A 都比 B 好。

最坏情况的渐进式复杂度往往最容易显现, 但它通常是最没用的。如果很容易确定所给定问题是哪一类, 通过对问题的特征分类来决定哪种算法更优是最有用的方法。不幸的是, 这种特征分类很难。

通过对问题的特征分类来确定哪种算法更优, 可以在理论上通过数学方法或在经验上通过建立程序来实现。这些数学定理的有效性只能依赖于作为基础的假设。相似地, 基于经验的程序验证只能在测试用例和算法实际的实现方法中有效。我们很容易通过举个反例来反驳一种算法对某类问题更优的猜想, 但却很难证明这种猜想。

如果第一个分支搜索就得到一个解,那么时间复杂度就与路径的深度呈线性关系,它只考虑这条路径上的元素,以及它的兄弟姐妹。最坏情况的时间复杂度是无限大。即使存在一个解,如果这个图是无限的或者循环的,深度优先搜索会陷入无限的分支然后永远找不到解。如果图是有限的树,前向分支系数最大是 b , 深度为 n , 那么最坏情况的时间复杂度是 $O(b^n)$ 。

【例 3-8】 考虑修改前面的图, 让 Agent 在位置之间移动更自由, 得出图 3-6, 一条无限的路径从 ts 指向 $mail$, 然后回到 ts , 再返回 $mail$, 如此循环往复。显然, 深度优先搜索将沿着这条路径永远搜索下去, 而不会考虑到 $b3$ 或者 $o109$ 等可选择路径。前 5 次反复使用深度优先路径搜索算法得到的边界如下:

```
[<o103>]
[<o103,ts>,<o103,b3>,<o103,o109>]
[<o103,ts,mail>,<o103,ts,o103>,<o103,b3>,<o103,o109>]
[<o103,ts,mail,ts>,<o103,ts,o103>,<o103,b3>,<o103,o109>]
[<o103,ts,mail,ts,mail>,<o103,ts,mail,ts,o103>,<o103,ts,o103>,<o103,b3>,<o103,o109>]
```

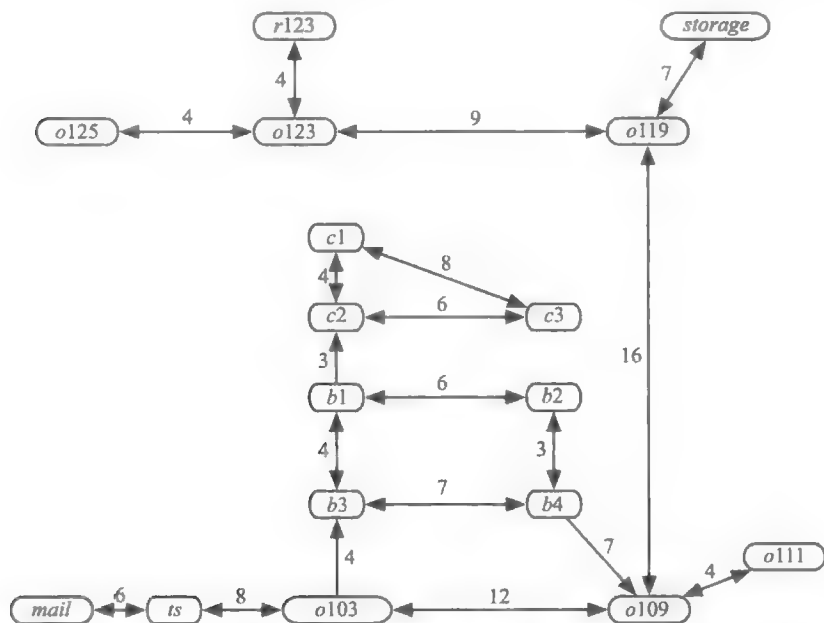


图 3-6 投递机器人域的有环图。 $X \leftrightarrow Y$ 表示从 X 到 Y 有一段弧, 从 Y 到 X 也有一段弧, 即 $\langle X, Y \rangle \in A$ 及 $\langle Y, X \rangle \in A$

可以通过不考虑有环路径来优化深度优先搜索。

因为深度优先搜索对邻居加入边界的顺序敏感, 这一点必须仔细对待。这个顺序可以是静态的(这样邻居的顺序固定)或者是动态的(邻居的顺序由目标节点决定)。

深度优先搜索适用的条件:

- 空间是被限定的;
- 存在许多解, 可能路径很长, 尤其当几乎所有路径都导致一个解时;
- 节点邻居的压栈的顺序可以调整, 以便在第一次尝试中就找到解。

深度优先搜索不适用的条件:

- 当图是无限的或者图中有环的时候, 有可能陷入无限的路径中;

- 解路径存在于隐蔽的深度，因为在这种情况下可能要搜索很多条长路径才能发现短路径结果。

深度优先搜索是许多其他搜索算法的基础，如迭代搜索。

3.5.2 宽度优先搜索

在**宽度优先搜索**(breadth-first search)中，边界是用一个先进先出(FIFO)队列实现的。因此，最开始从边界中选择的节点是最早进入队列的。

这个方法意味着从初始节点出发的路径按照弧数目多少的顺序依次产生。在每一个阶段中，选择弧最少的那条路径。

84

【例 3-9】来看图 3-7 中的树形图。假定起始节点在顶部。在宽度优先搜索中，节点扩展的顺序与目标节点的位置无关，这与深度优先搜索相同。最开始搜索的 16 个节点的扩展顺序已经在图中标记出来了。在最开始扩展的 16 步完成后，阴影节点就是多条路径的末端，并组成了边界。

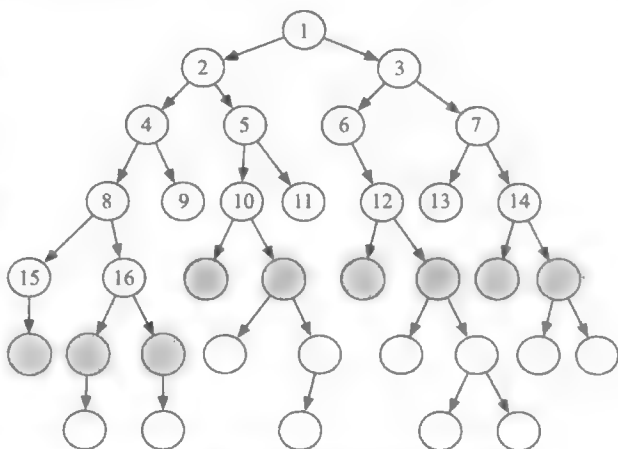


图 3-7 宽度优先搜索中节点的扩展顺序

【例 3-10】考虑图 3-2 从 o103 的宽度优先搜索。唯一的节点是 r123。最开始，边界是[<o103>]。然后经过 o103 的邻居得到扩展的边界，即[<o103, ts>, <o103, b3>, <o103, o109>]。这些都是 o103 的一条出弧所指向的节点。下面选择的三条路径是<o103, ts>, <o103, b3>, <o103, o109>，这个阶段，边界扩展为：[<o103, ts, mail>, <o103, b3, b1>, <o103, b3, b4>, <o103, o109, o111>, <o103, o109, o119>]。

这些路径都包含有两段弧，并都始于 o103。经过下一步的边界节点选择和扩展，得到新的五条路径，这时边界包括的从 o103 出发的路径都含有三段弧。记作：[<o103, b3, b1, c2>, <o103, b3, b1, b2>, <o103, b3, b4, o109>, <o103, o109, o119, storage>, <o103, o109, o119, o123>]。

注意到边界的每一条路径都有近似相等的弧数。对于宽度优先搜索，通常边界中路径的弧数最大相差 1 条。

85

假定搜索的分支系数是 b 。如果边界的第一条路径包含 n 段弧，那么边界中就至少有 b^{n-1} 个节点。所有的这些路径包含 n 或者 $n+1$ 段弧。因此，根据到达目标节点弧数最少的那条路径上的弧数，空间和时间复杂度呈指数级增长。无论如何，这个方法可以保证只要有解就能找到，并且是最少弧的解。

宽度优先搜索适用于：

- 不存在空间问题；
- 想找最少弧的路径；
- 可能有很少的解，但是至少有一个短路径解；
- 可能存在无限的路径，因为它会遍历所有的搜索空间，即使是无限的路径。

这个算法并不适用于所有的路径都很长，或者有一些可用的启发知识的情况。因为空

间复杂度很大, 所以它并不常用。

3.5.3 最低花费优先搜索

当路径的花费与弧有关时, 我们常常想找到总花费最小的路径。例如, 对于一个投递机器人, 花费可能是两位置之间的距离, 我们需要总距离最短的那条解路径。花费也可能是机器人按照弧实施动作所需要的各种资源。一个指导系统的花费可能是学生所需要的时间和精力。在每一种情况下, 搜索者都试图找到最低花费的解路径来达到目标。

目前为止考虑的搜索算法不能保证找到最低花费的路径, 他们完全没有使用弧花费的任何信息。宽度优先搜索首先找到弧数最少的路径, 但这未必是最少花费的路径。

最简单的能保证找到最低花费路径的搜索方法与宽度优先搜索算法相似。不同的是, 它是寻找最低花费的路径, 而不是通过扩展路径找最少弧数的路径。这是通过把边界作为一个由花费函数排序的优先级队列来实现的。

【例 3-11】 看图 3-2, 考虑图中从 $o103$ 开始的最低花费优先搜索。唯一的目标节点是 $r123$ 。这个例子中, 路径由最后的节点表示, 下标是路径的花费值。

最初, 边界是 $[o103_0]$ 。下一阶段是 $[b3_4, ts_8, o109_{12}]$ 。因为 $b3$ 的花费最小, 所以选择了到 $b3$ 的路径, 结果边界变为: $[b1_8, ts_8, b4_{11}, o109_{12}]$ 。

因为 $b1$ 的花费最小, 然后选择到 $b1$ 的路径, 结果为: $[ts_8, c2_{11}, b4_{11}, o109_{12}, b2_{14}]$ 。

因为 ts 的花费最小, 然后选择到 ts 的路径, 结果为: $[c2_{11}, b4_{11}, o109_{12}, mail_{14}, b2_{14}]$ 。

然后选择到 $c2$ 的路径, 如此循环。注意最低花费优先搜索是如何逐渐增长路径的, 它总是扩展花费值最低的路径。

如果一个问题存在解路径, 弧线的花费被界定为低于一个正常数且分支系数也是有限的, 那么最低花费优先搜索就能确定找到一个最优解(即有最低花费的路径)。而且, 最先找到的路径就是花费最低的路径。因为算法产生的路径从最初就是按照路径花费值排序的, 所以这个解是最优的。如果存在一个比第一个被发现的解更优的解, 它在边界中会被选择得更早。

界定的弧花费是用来保证最低花费搜索能找到最优解。如果没有这样的界定, 就会有有限花费的无限路径。例如, 对于每一个 $i > 0$, 节点 n_0, n_1, n_2, \dots 对应的弧线 $\langle n_{i-1}, n_i \rangle$ 中每一条弧线的花费都是 $1/2^i$, 这样的话, 存在很多形式为 $\langle n_0, n_1, n_2, \dots, n_k \rangle$ 的路径, 它们的花费都小于 1。如果有一条弧从 n_0 到目标节点的花费大于或等于 1, 它将永远都不会被选中。这个就是早在 2300 年前亚里士多德写到的“芝诺悖论”的基础。

像宽度优先搜索, 最低花费优先搜索在时间和空间上都是典型的指数函数。它会产生所有的低于解路径花费的从起始节点开始的路径。

3.6 启发式搜索

前面说的所有的算法都是无信息的, 并没有考虑目标节点在哪里。它们没有使用任何指引它们该向哪去的信息, 除非它们无意中发现目标。关于哪些节点是最有希望节点的启发信息的一种形式叫做启发函数 $h(n)$, 这个函数参数为节点 n , 返回一个非负实数, 即为节点 n 到目标节点的估计花费。如果 $h(n)$ 的值小于或等于从 n 节点到目标节点最低花费路

径的实际花费,那么就说启发函数是“低估”的。

启发函数是得出目标的搜索方向的一种方式。它提供一个有信息依据的方法来猜测节点的哪个邻居将会通往目标节点。

启发函数没有什么神奇可言。它只能使用可容易地从节点上获取的信息。通常情况下,需要在以下两个值之间做折中:获取一个节点的启发值所花费的工作量,以及一个节点的启发值如何准确地衡量从该节点到目标的实际路径花费。

得到启发函数的一个标准方式是:解决一个更简单点的问题,然后将简化了的问题的实际花费作为原始问题的启发函数。

【例 3-12】 图 3-2 中,从节点到目标位置的直线距离可以作为启发函数。

下述实例中假设如下启发函数:

$h(mail) = 26$	$h(ts) = 23$	$h(o103) = 21$
$h(o109) = 24$	$h(o111) = 27$	$h(o119) = 11$
$h(o123) = 4$	$h(o125) = 6$	$h(r123) = 0$
$h(b1) = 13$	$h(b2) = 15$	$h(b3) = 17$
$h(b4) = 18$	$h(c1) = 6$	$h(c2) = 10$
$h(c3) = 12$	$h(storage) = 12$	

h 函数就是一个低估,因为 h 的值小于或等于从节点到目标的最低花费路径的真实花费。对于 $o123$ 来说, h 的值就是真实花费值。对 $b1$ 来说,则是很大的低估,看起来 $b1$ 离目标很近,但其实只能通过一条很长的路径才能到达目标。而对于 $c1$ 来说,这个 h 值则差之千里,因为看起来它离目标节点很近,事实上没有从该节点通往目标节点的路径。◀

【例 3-13】 在例 3-2 中的投递机器人,状态空间包含要运送的包裹。假定花费函数是机器人运送所有包裹走过的总距离。一个可能的启发函数就是一个包裹到目的地的最大距离。如果机器人只能携带一个包裹,那么可能的启发函数就是包裹经过的距离之和。如果机器人一次可以携带多个包裹,那么这个启发函数就不是真实花费的低估了。◀

h 函数也可以扩展适用(非空)路径。路径的启发值是路径最后的节点的启发值。即

$h(\langle n_0, \dots, n_k \rangle) = h(n_k)$

启发函数的一个简单应用是在深度优先搜索中为邻居排序,这些节点被依次压入堆栈来表示边界。这样邻居被加入边界,因此最先选择的是最优邻居,即所谓“启发式深度优先”(heuristic depth-first search)算法。这种搜索算法可以找到局部最优路径,但是在搜索另一条路径之前,它会遍历所有已选路径。尽管这个算法常用,但也受困于深度优先搜索自身的问题。

启发函数的另一种使用方式是:总是选择边界中具有最低启发值的路径,这叫做**最优优先搜索**(best-first search)。这个方法通常不太有效,它会选择看上去最有希望的路径,因为他们离目标节点很近,但是路径花费会不断增加。

【例 3-14】 如图 3-8 所示,弧线的花费是它的长度。目标是找到从 s 到 g 的最短路径。假定到节点 g 的欧几里得距离是启发函数。启发深度优先搜索将会选择位于 s 下面的节点,并永远不会终止。相似地,因为位于 s 以下的所有节点看起来都不错,最优优先搜索将会在它们之间循环,而从不尝试由 s 开始的其他可替换路径。

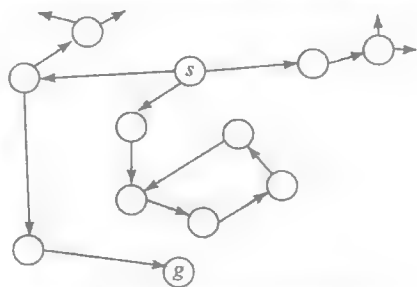


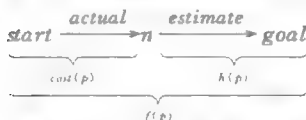
图 3-8 不适用最优优先搜索的图

3.6.1 A* 搜索

A* 搜索是最低花费优先搜索和最优优先搜索的结合，它在选择路径扩展时既要考虑路径的花费也要考虑启发信息。对于边界上的每一条路径，A* 都会估计沿路径从初始节点到目标节点的总路径花费。它使用函数 $cost(p)$ (即已找到的路径花费) 和启发函数 $h(p)$ (即从 p 的结尾到目标的估计花费)。

对于边界上的任意路径 p ，定义函数 $f(p) = cost(p) + h(p)$ ，指沿着路径 p 到目标节点的估计总花费。

如果 n 是路径 p 的末节点，那么可以描述为：



如果 $h(n)$ 是从节点 n 到目标节点路径花费的低估，那么 $f(p)$ 就是从初始节点经过 p 到目标节点路径花费的低估。

A* 是通过将边界看做按照 $f(p)$ 进行排序的优先级队列来实现的。

【例 3-15】 对例 3-4 运用 A* 算法进行搜索，并使用例 3-12 的启发函数。在这个例子中，边界上的路径用路径上最后一个节点来表示，下标是路径的 f 值。初始边界是： $[o103_{21}]$ ，因为 $h(o103) = 21$ ，路径花费是 0。然后用邻居替换，形成新的边界： $[b3_{21}, ts_{31}, o109_{36}]$ 。

第一个元素代表路径 $\langle o103, b3 \rangle$ ，它的 f 的值为： $f(\langle o103, b3 \rangle) = cost(\langle o103, b3 \rangle) + h(b3) = 4 + 17 = 21$ 。接着 $b3$ 就被邻居替换，边界变为： $[b1_{21}, b4_{29}, ts_{31}, o109_{36}]$ 。

然后到 $b1$ 的路径被 $b1$ 的邻居替换，边界变为： $[c2_{21}, b4_{29}, b2_{29}, ts_{31}, o109_{36}]$ 。

接下来，到 $c2$ 的路径被它的邻居替换，边界变为： $[c1_{21}, b4_{29}, b2_{29}, c3_{29}, ts_{31}, o109_{36}]$ 。

直到这个状态，搜索一直都在找看起来直接通往目标节点的路径。下面，选择 $c1$ 被相邻点替代，形成边界： $[b4_{29}, b2_{29}, c3_{29}, ts_{31}, c3_{35}, o109_{36}]$ 。

直到这一步，到 $c3$ 有两条不同路径。其中一条不经过 $c1$ ，它的 f 值要比经过 $c1$ 的另外一条路径低。在后面，我们会考虑将其中一个路径剪枝的情况。

这里两条相同 f 值的路径。算法没有指定选择哪条。假设下一步选择了通往 $b4$ 的路径，并被它的相邻点替换，形成边界： $[b2_{29}, c3_{29}, ts_{31}, c3_{35}, o109_{36}, o109_{42}]$ 。

然后选择到 $b2$ 的路径并被邻居替换，但这条路径是空集，所以形成边界： $[c3_{29}, ts_{31}, c3_{35}, b4_{35}, o109_{36}, o109_{42}]$ 。

然后到 $c3$ 的路径被删除，它没有邻居，因此，新的边界变为： $[ts_{31}, c3_{35}, b4_{35}, o109_{36}, o109_{42}]$ 。

注意 A* 算法是如何从一开始寻求多个不同路径的。

最终会找到一个最低花费路径。这个算法会强制尝试很多不同的路径，因为它们其中的一些路径暂时看上去有最低花费。它仍然比最低花费优先和最优优先搜索方法都要好。

【例 3-16】 考虑图 3-8，对于其他的启发方法，这是个有问题的图。虽然由于启发函数，起始情况从节点 s 向下搜索，但最终路径的花费变得非常大，因此它选择了在实际最优路径上的节点。

如果解存在，A* 总是能找到最优解，而且搜索的第一条路径就是最优的，这个性质

叫做 A^* 的“可采纳性”(admissibility)。它意味着：即使当搜索空间是无限的，如果解存在，那么就一定能找到解，而且找到的第一条路径就是最优解，即从起始节点到目标节点的最低花费路径。

命题 3.1(A^* 可采纳性) 如果存在解， A^* 总可以找一个解，并且找到的第一个解就是最优解，如果：

- 分支系数有限(每一个节点有有限个邻居)；
- 弧线花费大于 $\epsilon > 0$ ；
- $h(n)$ 是从节点 n 到目标节点的最低花费路径的最小实际花费值的下界。

证明：

A 部分：可以找到解。如果弧线的花费都大于 ϵ , $\epsilon > 0$ ，最终对于边界中所有的路径 p , $cost(p)$ 会超过任何有限的数。因此，如果存在解， $cost(p)$ 会超过解的花费(搜索树的深度不会大于 m/ϵ ，这里 m 是解的花费)。因为分支系数是有限的，在搜索树还没扩展到这个规模之前，只能扩展有限数目的节点，但是那时 A^* 算法已经找到了解路径。

B 部分：找到的第一条解路径就是最优路径。在最优解路径中，任何一个节点的 f 值都小于或者等于最优解的 f 值。这是因为 h 是从一个节点到目标的实际花费的低估。因此，最优解路径中任何一个节点的 f 值都小于任何非最优解的 f 值。所以，当一个节点存在于指向最优解的边界中时，算法永远不会选择一个非最优的路径(因为在每一步中，都会选择 f 值最小的元素)。因此，在可能选择一个非最优解之前，它肯定已经造访了一个最优路径上的所有的节点，包括每一个最优解。 ■

应该注意到： A^* 的可采纳性不能保证每一个从边界中选择的中间节点位于从开始节点到目标节点的最优路径中。可采纳性使算法不必担心环路问题，保证了第一个解是最优解。但当局部路径最优时，它并不能保证算法不会改变主意。

看一下启发函数是如何提高 A^* 算法的效率的，假设 c 是从起始节点到目标节点的最短路径的花费。 A^* 具有可采纳性的启发信息，扩展集合 $\{p: cost(p) + h(p) < c\}$ 中从起始节点开始的每一条路径，或者集合 $\{p: cost(p) + h(p) = c\}$ 中的某些路径。

91

如果减少这些集合中第一部分的数量，提高 h 则会影响 A^* 的效率。

3.6.2 搜索策略总结

图 3-9 中的表格给出了各种搜索策略的总结。

策略	从边界选择节点	终止否	空间
深度优先搜索	加入上一个节点	No	线性
宽度优先搜索	加入第一个节点	Yes	指数级
最优优先搜索	全局启发值最小 $h(p)$	No	指数级
最低花费优先搜索	花费最小 $cost(p)$	Yes	指数级
A^*	启发值和花费之和最小 $cost(p) + h(p)$	Yes	指数级

图 3-9 搜索策略的总结

注：“终止否”意味着“如果图(可能无穷大)中有一条到目标的路径，该路径上每个节点都有有限个数量的邻居，且每条弧的花费都有正值下界，那么这个方法能否保证搜索终止？”。那些回答“Yes”的搜索策略的最坏情况下的时间复杂度与路径长度呈指数形式增长，那些不能保证终止的算法有无限最坏情况的时间复杂度。“空间”指的是空间复杂度，它与路径长度是“线性”或是“指数级”关系。

深度优先搜索在空间上与造访路径的长度呈线性增长，但是即使有解存在，它也不能保证能找到解。宽度优先、最低花费优先以及 A^* 算法在时间和空间上都是呈指数增长，只要解存在，即使图是无限大的(只要分支系数有界，弧线花费是正非平凡)，它们都能保

证找到解。

最低花费优先和 A^* 搜索算法都能保证第一个找到的解是最低花费的解。

3.7 更复杂的搜索方法

之前的策略可以进行很多优化。首先，我们给出两种适用于图中有环的方法：一种是对环路的明确检查，而另一种是对到某一节点的多路径检查。然后，我们给出迭代深度算法和深度分支界限搜索法，这两种方法是能保证找到解路径（甚至是最优解）的一般方法，就像宽度优先搜索算法或者 A^* 搜索算法，但是利用了深度优先搜索的空间优点。我们将搜索问题分解为很多小的搜索问题来简化搜索方法，这样每一个都更容易解决。最后，我们说明利用动态规划法如何寻找路径和构建启发函数。

3.7.1 环检查

用来代表搜索空间的图可能包含环。例如，图 3-6 中的机器人域中的机器人会在节点 $o103$ 和 $o109$ 之间来回移动。前面所述的搜索方法中有些会陷入环，不断循环，甚至在有限图中也找不到搜索结果。其他的方法也可能循环，但最终会找到解。

当能保证在有限图中找到解后，修剪搜索树的最简单的方法是，不考虑那些已经在从起始节点开始的路径上的点。“环检查”(cycle check)和“回路检查”(loop check)就是用来检查最后一个节点是否在从初始节点到这个点的路径上出现过。有了循环检查，只有路径 $\langle s, \dots, s_k, s \rangle$ ，其中， $s \in \{s_0, \dots, s_k\}$ ，加入了图 3-4 中第 16 行。另外，也可以在选择节点之后再进行检查，删除有环的路径。

环检查的计算复杂性决定于所使用的搜索方法。对于深度优先搜索方法，其图是显式存储的，管理费用跟常数因子一样低。图中每个节点增加 1 位，当节点扩张的时候该位应当为 1，回溯的时候为 0。搜索算法可以通过不扩展该位是 1 的节点来避免循环。因为深度优先算法保持单一当前路径，所以这个方法有效，路径上的边界有可替代的分支。即使图是动态生成的，只要当前路径上的点有一个有效的索引结构，那么很容易完成环检查。

对于多路径的搜索策略（也就是说，图 3-9 中所有有指数空间的那些），环检查的时间与搜索的路径长度保持线性增长。在检查以保证不会添加已经在路径中出现过的节点上，这些算法不能比搜索局部路径做得更好。

3.7.2 多路径剪枝

通常有多条路径通向一个节点。如果只需要一条路径，对于已经找到一条路径的节点，搜索算法就需要从边界中剪枝掉其他任何通往该节点的路径。

多路径剪枝(multiple-path pruning)是通过由已扩展节点组成的 **closed** 表来实现的。如图 3-4 中的 13 行，选择了一个路径，如果它的最后一个节点在 **closed** 表中，那么这条途径就该删除。否则，将最后的节点加到 **closed** 表中，算法会继续进行。

这个方法不一定保证最短的路径不被丢弃。可能要更复杂的方法才能保证找到最优解。为了保证搜索算法可以找到至目标节点最低花费的路径，应当遵守以下几条之一：

- 保证到达任何节点的第一条路径都是到那个点的最低花费路径，然后像前面讨论过的那样，剪枝掉所有随后找到的到那个点的路径。

- 如果搜索算法找到比已经找到的路径花费更低的路径了,那么它就可以将所有较高花费的路径删除(因为这些都不可能是最优解)。也就是说,如果边界有一条路径 p 在路径 $\langle s, \dots, n, \dots, m \rangle$ 中,当发现到 n 的路径 p' 比 p 中相应部分更近,那么 p 就可以从边界中删除了。
- 无论何时,如果搜索中找到比之前找到的路径花费更低的至某点的路径,那么就这条新路径吸收到已扩展路径的初始路径中。因此,如果有路径 $p = \langle s, \dots, n, \dots, m \rangle$, 一个到 n 的更短的路径 p' 就可以代替到 n 的路径 p 的初始部分。

这些方案中的第一条,允许在找到最优解的保证下使用 closed 表。其他则需要更复杂的算法。

在最低花费优先算法中,第一次找到的到某一个节点(如,从边界中选择一个节点)的路径,是到该节点的最低花费路径。删除随后的到该节点的路径不会移除最低花费的那条到该节点的路径,所以,剪掉随后的路径仍然可以保证找到最优解。

像之前描述的一样, A^* 算法不能保证到某点第一次选择的路径就是到该点的最低花费路径。注意到,可采纳性定理保证的是对于到目标节点的每一条路径,而不是每一条路径。看看何时删除到一个节点的随后路径会删除最优解,假设算法选择了到节点 n 的路径 p 来扩展,但是存在一个到节点 n 的更低花费的路径,而这条路径还没有找到。那么,边界中就应该有路径 p' ,它是较低花费路径的初始部分。猜想路径 p' 在节点 n' 结束。则一定有 $f(p) \leq f(p')$, 因为 p 是在 p' 之前选择的。这就是说:

$$\text{cost}(p) + h(n) \leq \text{cost}(p') + h(n')$$

如果经过 p' 比经过 p 的到 n 的路径有更低花费,则

$$\text{cost}(p') + d(n', n) < \text{cost}(p)$$

其中, $d(n', n)$ 为从节点 n' 到节点 n 的最短路径的实际花费。从上面两个等式,我们可以得出:

$$d(n', n) < \text{cost}(p) - \text{cost}(p') \leq h(p') - h(p) = h(n') - h(n)$$

($h(p') - h(p) = h(n') - h(n)$ 可通过以上假设得出。)因此,对于任意两个节点 n 和 n' , 如果 $|h(n') - h(n)| \leq d(n', n)$, 那么就能保证找到的第一条路径就是最低花费路径。对于任意两个节点 n 和 n' , 对 h 的单调限制(monotone restriction)是 $|h(n') - h(n)| \leq d(n', n)$ 。也就是说,两个节点的启发函数值的差必须小于或等于两点最低花费路径的实际花费。例如,当花费函数是关于距离的函数时,它适用于两点之间的欧几里得距离(在 n 维欧几里得空间的直线距离)的启发函数。它也适用于启发函数是有更短解的简化问题的解。

因为单调的限制,在边界, f 值是单调非递减函数。也就是说,随着边界的扩展, f 值不会变小。因此,因为单调的限制, A^* 算法中随后的到任何节点的路径都可以删除。

多路径剪枝包含环检查,因为环是到某个节点的另外一条路径,因此要被删除掉。如果图是显式存储的,通过在每个节点上设置一个比特位标明已被找到的路径,多路径剪枝可以在一个常数时间内完成。如果图是动态生成的,通过存储由已扩展过的所有节点组成的 closed 列表,它能在对数时间(只要进行适当索引,可与已扩展的节点数量成对数级)完成。多路径剪枝优先于宽度优先的环检查,因为宽度优先搜索中所有节点被认为必须存储。然而,对于深度优先搜索,算法并不是存储所有的已经扩展节点。存储它们会使这个方法的空间呈指数增长。因此,对于深度优先搜索,环检查优先于多路径剪枝。

3.7.3 迭代深化

到目前为止,我们所讨论过的方法中,没有一种方法是理想的。仅有的能保证找到路径方法需要指数空间(见图 3-9)。结合深度优先搜索算法的空间效率和宽度优先搜索算法的最优性的一种方法是**迭代深化**(iterative deepening)。这种方法的思想是重新计算边界中的点而不是储存它们。每一次重新计算的过程都是深度优先搜索,因此用更少的空间。

我们考虑将宽度优先搜索融合到迭代深化搜索。由仅进行有限深度搜索的搜索器来进行。首先,通过用深度为 1 的方式进行深度优先搜索,找到深度为 1 的深度优先方式的路径。然后找深度为 2 的路径,然后是深度为 3 的路径,依次进行。每一次重新计算都可以删掉之前的计算并再次启动。最后如果解存在就可以找到解了,因为它是按顺序列举路径,所以第一个找到的路径就是弧线最少的。

当使用迭代深化搜索时,应当注意区分:

- 因为达到深度界限的失败。
- 还未到达深度界限的失败。

对于第一种情况,搜索必须进行更深界限的搜索。对于第二种情况,重新进行一次更深界限搜索是浪费时间,因为不管多深,路径都不存在。我们称这种到达了深度界限的失败叫做**非自然失败**(failing unnaturally),未到达深度界限的失败叫做**自然失败**(failing naturally)。

图 3-10 中提出一种迭代深化搜索的实现 *IdSearch*。局部程序 *dbsearch* 实现了一个深度有界的深度优先搜索(使用递归来保持堆栈),可以限制正在搜索的路径长度。它使用深度优先搜索来找到所有长度为 $k+b$ 的路径,其中, k 是给定的从起点开始的路径长度, b 是一个非负整数。迭代深化搜索器调用这个程序来增加深度界限。这个程序发现的到目标节点的路径的顺序和宽度优先算法相同。作为一种通用的图搜索算法,为找到更多路径,在 22 行的 *return* 之后,搜索能从 23 行继续。

只要宽度优先搜索失败,迭代深化搜索就会失败。当需要更多解时,即使在后续的迭代中可能被再次发现,每一个成功路径也只返回一次。通过跟踪何时增加界限可以帮助找到解能使搜索终止:

- 如果深度搜索中因为到达了深度界限而截断了,那么深度界限可以增加。在这种情况下,搜索就非自然失败了。如果因为深度界限的限制而没有删除任何路径,那么搜索就自然失败了。这种情况下,程序终止,报告 0 路径。

```

1: procedure IdSearch(G, s, goal)
2:   Inputs
3:     G: 具有 N 个节点和 A 条边的图
4:     s: 开始节点的集合
5:     goal: 关于状态的布尔函数
6:   Output
7:     从 s 到 goal 函数值为真的节点的路径
8:     或者如果没有这样的路径则⊥
9:   Local
10:    natural_failure: 布尔型
11:    bound: 整型
12:    procedure dbsearch( $\langle n_0, \dots, n_k \rangle$ , b)
13:      Inputs
14:         $\langle n_0, \dots, n_k \rangle$ : path
15:        b: integer,  $b \geq 0$ 
16:      Output
17:        到目标节点的长度为  $k+b$  的路径
18:      if  $b > 0$  then
19:        for each arc( $n_k, n$ )  $\in A$  do
20:          dbsearch( $\langle n_0, \dots, n_k, n \rangle$ ,  $b-1$ )
21:      else if goal( $n_k$ ) then
22:        return  $\langle n_0, \dots, n_k \rangle$ 
23:      else if  $n_k$  有任意邻居 then
24:        natural_failure  $\leftarrow$  false
25:      bound  $\leftarrow$  0
26:      repeat
27:        natural_failure  $\leftarrow$  true
28:        dbsearch( $\{ \langle s \rangle : s \in S \}$ , bound)
29:        bound  $\leftarrow$  bound + 1
30:      until natural_failure
31:      return ⊥

```

图 3-10 迭代深化搜索

- 搜索只报告在之前的迭代中没有报告过的一条路径，因此，它只返回那些长度等于深度界限的路径。

迭代深化搜索的最明显的问题是每一步的计算浪费。然而，这可能不像某些人想的那样糟糕，尤其是当分支系数很高的时候。考虑下面算法的运行时间。假定一个常数分支系数 $b > 1$ ，考虑界限是 k 的搜索。对于深度 k ，有 b^k 个节点，每一个节点被生成 1 次。深度为 $k-1$ 的节点被生成 2 次，那些深度为 $k-2$ 的节点被生成 3 次，如此类推，然后深度为 1 的节点被产生 k 次。因此，节点的生成总数是：

$$\begin{aligned} & b^k + 2b^{k-1} + 3b^{k-2} + \dots + kb \\ &= b^k (1 + 2b^{-1} + 3b^{-2} + \dots + kb^{1-k}) \\ &\leq b^k \left(\sum_{i=1}^{\infty} ib^{i(1-b)} \right) \\ &= b^k \left(\frac{b}{b-1} \right)^2 \end{aligned}$$

在深度 n 生成节点有一个固定的开销 $(b/(b-1))^2$ ，当 $b=2$ 时，就有一个开销系数 4，当 $b=3$ 时，有开销系数 2.25 保证边界。该算法是 $O(b^k)$ ，并且找不到一个更好的渐进的无信息搜索策略。注意到，如果分支系数接近 1，这种分析就没有用（因为分母趋近于 0），见习题 3.9。

迭代深化也可以用于 A^* 搜索中。迭代深化 A^* (IDA*) 算法通过重复的有深度界限的深度优先搜索来进行。它是一个有界的 $f(n)$ 的值，而不是路径中弧的数量有界。它的阈值起始于是 $f(s)$ ，其中 s 是有最小 h 值的初始节点。然后 IDA* 进行深度优先的深度有界搜索，但从不扩展一个比当前的界限更高的 f 值的节点。如果深度有界的搜索非自然地失败了，那么下一个界限就是超过之前界限的最小的 f 值。IDA* 就如 A^* 一样检查相同的节点，但用深度优先搜索重新计算它们，而不是存储它们。

3.7.4 分支界限法

深度优先分支界限(branch and bound)搜索是一种结合了深度优先搜索算法的空间节约和启发信息搜索的方法。它特别适用于存在很多条通往目标节点的路径，我们需要的是最优的路径。像在 A^* 搜索中，我们假设 $h(n)$ 小于或等于从 n 到目标节点的最低花费路径的花费。

深度优先分支界限搜索的思想是保持到目标最低花费的路径和它的花费，假定这个花费是有界的。如果搜索碰到路径 p 满足 $\text{cost}(p) + h(p) \geq \text{bound}$ ，那么 p 就该删除掉。如果找到没被删除的路径，那么这条路径一定优于之前最好的路径。这个新解被记住，而且界限就设置成这个新解的花费，然后寻找更好的解。

分支界限搜索生成持续优化解的序列。一旦找到一个解，就一直优化它。

分支界限搜索典型地用于深度优先搜索，这样可以保持深度优先的节省空间的优点。它和深度有界搜索的实现方法相似，但是这里的界限是路径的花费，然后减少花费发现更短的路径。算法会记住所发现的花费最低的路径，当搜索完成时返回这条路径。

算法见图 3-11。对于花费有界搜索，内部程序 *cbsearch* 使用全局变量来给主程序提供信息。

最初，界限可以设定为无穷大，但是也可以设置一个高估的值， bound_0 ，作为最优路径的花费，这很有用。如果存在比初始花费界限 bound_0 更低的路径，这个算法会返回最优解（从初始节点到目标节点的最低花费路径）。

```

1: procedure DFBranchAndBound( $G, s, goal, h, bound_0$ )
2:   Inputs
3:      $G$ : 具有  $N$  个节点和  $A$  条边的图
4:      $s$ : 开始节点
5:      $goal$ : 节点上的布尔函数
6:      $h$ : 节点上的启发函数
7:      $bound_0$ : 初始化深度(如果没有指定, 可以是  $\infty$ )
8:   Output
9:     如果没有一个含有  $c$  的解小于  $bound_0$ , 返回从  $s$  到目标节点的最低花费的路径
10:    或者如果没有解小于  $bound_0$ , 则  $\perp$ 
11:   Local
12:      $best\_path$ : 路径或  $\perp$ 
13:      $bound$ : 非负实数
14:     procedure  $cbsearch(\langle n_0, \dots, n_k \rangle)$ 
15:       if  $cost(\langle n_0, \dots, n_k \rangle) + h(n_k) < bound$  then
16:         if  $goal(n_k)$  then
17:            $best\_path := \langle n_0, \dots, n_k \rangle$ 
18:            $bound := cost(\langle n_0, \dots, n_k \rangle)$ 
19:         else
20:           for each arc  $\langle n_i, n \rangle \in A$  do
21:              $cbsearch(\langle n_0, \dots, n_k, n \rangle)$ 
22:      $best\_path := \perp$ 
23:      $bound := bound_0$ 
24:      $cbsearch(\langle s \rangle)$ 
25:   return  $best\_path$ 

```

图 3-11 深度优先分支限界搜索

如果初始界限略高于最低花费路径的花费, 那么这个算法不需扩展比 A^* 算法更多的弧就可以找到最优的路径。上述情况发生的条件是, 算法将那些花费比最低花费高的任何路径都剪枝; 一旦它发现一个通往目标节点的路径, 它就只探索那些 f 值低于已扩展路径的路径。而这些正好就是 A^* 找到一个解后所探索的路径。

当 $bound_0 = \infty$ 时, 如果返回 \perp , 则不存在解。当 $bound_0$ 是有界值时, 返回 \perp , 表示比 $bound_0$ 花费少的解不存在。在找到解或者已知解不存在之前, 这个算法可以结合迭代深化算法来增加界限, 见习题 3.13。

【例 3-17】 看图 3-12 中的树形图。目标节点被涂上阴影。假设每一段弧线的长度为 1, 没有启发信息(即每个节点的 $h(n)=0$)。在算法中, 假设 $depth_0 = \infty$, 深度优先搜索总是先选择最左边的后继节点。该图说明了检验确定节点是否是目标节点的顺序。没有标出序号的节点是未检验的节点。

在节点 5 下面的子树形图中, 没有目标节点, 被完全探索完(或者根据深度的有限值 $depth_0$ 进行扩展)。被检验的第 9 个节点是目标节点。它有一条路径, 其花费是 5, 所以界限值设为 5。从此以后, 只有长度小于 5 的路径才能作为可能的解被检验。第 15 个被

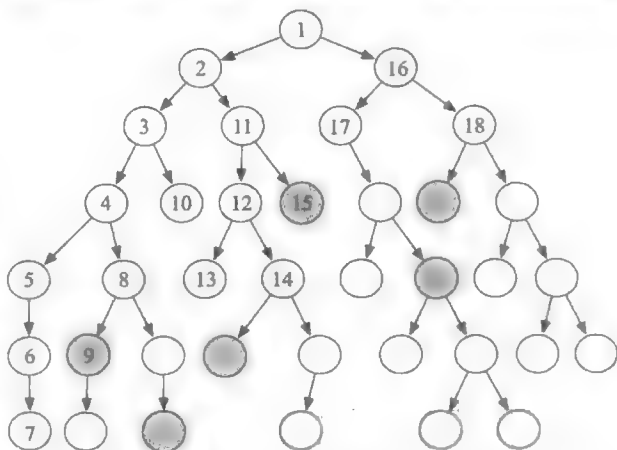


图 3-12 在深度优先分支限界搜索中所扩展的节点

检验的节点也是目标节点。它的路径花费是 3，所以界限减少到 3。再没有找到其他的目标节点，所以到标号 15 的那条路径返回，这个是最优路径。有另外一条最优路径被剪枝。算法就不再检验标号为 18 的节点的后继节点。

如果有启发信息，就可以用来剪枝部分的搜索空间，就像 A* 算法。

3.7.5 搜索方向

在有限无信息图上的通用搜索算法的搜索空间大小是 b^k ，这里 b 是分支系数， k 是路径长度。任何可以减少 b 和 k 的方法都可能会节省很多资源。

问题求解的图搜索方法的抽象定义在以下意义上是对称的，即这个算法可以从初始节点开始，向前搜索目标节点；或者逆向从目标节点开始，向后搜索初始节点。注意，目标节点的许多应用是由布尔函数隐式地确定，即当发现目标节点时返回 *true*，而不是显式地返回节点集，所以说逆向的向后搜索也许是不可能的。

对于那些目标节点是显式的情况，在一个方向上搜索可能比另一个方向更有效。搜索空间的大小与分支系数成指数增长。通常情况下，向前和向后搜索有不同的分支系数。向前或者向后搜索的一般原则是哪个方向有较小的分支系数。

下面讨论的几种方法对于搜索空间来说可以提高效率。

1. 双向搜索

这个搜索方法的思想是同时从起始节点向后以及从目标节点向前搜索来减少搜索时间。当两个搜索的边界相交，这个算法就重构出一个路径，它从起始节点到相交的边界再到目标节点。

在双向搜索中产生一个新问题，即要保证两个搜索的边界真正相交。举例来说，在两个方向上的深度优先搜索不一定能够很好地起作用，因为它可能会因为其较小的搜索边界而相互错过相交。在两个方向上的宽度优先搜索可以保证相交。

一个方向深度优先搜索结合另一个方向宽度优先搜索可以保证所需要的搜索边界的相交，但是在哪个方向上选择用哪种方法很困难。这取决于宽度优先节省的花费以及搜索检查深度优先搜索何时与其元素相交的花费。

在有些情况下，双向搜索可以节省很多。例如，如果前向分支系数和后向分支系数都是 b ，目标节点的深度是 k ，那么宽度优先搜索花费的时间与 b^k 成比例，然而一个对称双向搜索花费的时间与 $2b^{k/2}$ 成比例。即使时间复杂度仍然是指数级，它以指数函数形式节省时间。注意，这种复杂性分析需要假设找到相交点是无需代价的，在很多应用中这种假设可能不成立(见习题 3.10)。

2. 岛驱动搜索

一种使搜索更有效的方式是找到有限的向前和向后搜索可以相交的地方，例如，找到不同楼层的两间房间之间的通路，合适的搜索限制可能是先去其中一个楼层的电梯，然后到目标楼层的那个电梯。直觉上，这些指定的位置在搜索图上就像一些岛(island)，它们被限定在从起始节点到目标节点的解路径上。

当岛确定了，Agent 可以将一个搜索问题分解为几个子问题，例如，一个从初始房间到电梯，一个从某一层的电梯到另外一层的电梯，一个从电梯到目标房间。通过三个更简单的问题求解就缩小了搜索空间。这些更小问题减少了大搜索的组合爆炸，这也是个很好的例子，说明了问题的额外信息如何提高搜索的效率。

为使用岛来找到 s 和 g 之间的一条路径，需要：

- 1) 确定一系列的岛 i_0, \dots, i_k ;
- 2) 找到从 s 到 i_0 , 从 i_{j-1} 到 i_j (j 为 $1 \sim k$), 从 i_k 到 g 的路径。

每一个搜索问题应该比总的问题更简单, 因此更容易解决。

岛的识别可能是我们从图中无法获得的额外信息。使用不适当的岛可能使问题更复杂 (甚至不可解决)。通过选择不同岛集, 我们可以将问题分解为另外一组子问题, 并通过可能的岛空间进行搜索。在实际使用中是否适用取决于问题的细节。

3. 抽象层次搜索

岛的概念可以用来定义在细节上的多个层次或者抽象的多个层次上所使用的问题求解策略。

抽象层次上的搜索的思想首先涉及问题的抽象, 尽可能去掉细节。我们可能找到问题的部分解: 它的解需要进一步的细节。例如, 从一个房间到另一个房间的问题, 需要使用多次转身, 但是 Agent 喜欢从一个抽象层次推理问题, 其中实际转向的细节被省略了。我们期望适当的抽象解决能从广义角度解决问题, 只剩下一些小问题还需要解决。像传递机器人的路径规划问题, 不省略细节的搜索求解很困难, 除非必须考虑它们。

这种方法的实现可以通过岛驱动搜索找到可能的岛。一旦在岛层次上找到一个解, 子问题就可以同样的方式通过递归解决了。在低层次发现的信息可以通知高层次, 其中的某些可能解不如期望的那么好。高层次可以使用这些信息重新规划, 这个过程通常不能保证最优解, 因为它只考虑一些高层次的分解。

在抽象层次的搜索很大程度上取决于怎样分解和抽象要解决的问题。一旦问题被抽象和分解出来, 任何搜索方法都能用来解决它们。然而, 识别有用的抽象和问题分解并不容易。

3.7.6 动态规划法

动态规划法是最优化的一般方法, 它可以存储问题的部分解, 所以对于已经发现的解可以检索而不必重新计算。动态规划算法的使用贯穿人工智能。

动态规划算法可以用来在图中找到路径。直觉上, 图搜索中的动态规划(dynamic programming)可以被看做用来构造完美的启发函数使得 A^* 可以保证找到解, 即使它只有边界的一个元素。 $cost_to_goal$ 函数表示每个节点到目标节点的最低花费路径的准确花费。

政策(policy)用来指定从每个节点引出哪条弧。 $cost_to_goal$ 函数可以离线计算, 用来建立最优的政策。Agent 可以利用这个政策在线决定在这个节点做什么。

令 $cost_to_goal(n)$ 是从 n 到目标节点的最低花费路径的实际花费。 $cost_to_goal(n)$ 函数可以定义为:

$$cost_to_goal(n) = \begin{cases} 0 & \text{如果是 } is_goal(n) \\ \min_{(n,m) \in A} (cost(\langle n,m \rangle) + cost_to_goal(m)) & \text{其他} \end{cases}$$

一般的思想是从目标节点开始, 建立一个表存放每一个节点的 $cost_to_goal(n)$ 值。这可以通过多路径剪枝的最低花费优先搜索来实现, 它从目标节点开始在反向图(将图中所有弧的方向反过来)中搜索。对于已发现的每个节点, 动态规划算法记录其 $cost_to_goal$ 值, 而不是有一个需搜索的目标。使用反向图计算每个节点到目标节点的花费, 而不是从目标到每个节点。实质上, 动态规划从目标节点进行逆向搜索, 它通过建立从图中每个节点到目标节点的最低花费路径来完成。

对于一个特定的目标, 一旦每个节点的 $cost_to_goal$ 值记录下来, Agent 就可以根据

这个值来决定最优路径的下一段弧。从节点 n 应该选择这样的邻居 m ，它有 $cost(\langle n, m \rangle) + cost_to_goal(m)$ 的最小值。遵循这个政策，Agent 就能沿着最低花费路径从任意节点直到目标。假设每个节点有有限邻居，给定 $cost_to_goal$ ，决定哪段弧是最优的需要常数时间，该常数与图的大小有关。动态规划法建立 $cost_to_goal$ 表所需要的时间和空间与图的大小呈线性关系。

103

动态规划算法在下面条件下是有用的：

- 目标节点是显式的(之前的方法只假设一个函数来识别目标节点)；
- 需要一个最低花费路径；
- 图是有限的和足够小的，能够存储每一个节点的 $cost_to_goal$ 值；
- 目标不经常变化；
- 政策对于每个目标可以使用多次，产生 $cost_to_goal$ 值的花费可以分摊到很多实例问题上。

动态规划搜索的主要问题是：

- 只有当图有限，而且表能小到适应于内存时，这个方法才有效。
- 对于每一个不同的目标，Agent 必须重新计算政策。
- 需要的时间空间线性于图的大小，对于有限的图来说，图的大小通常是路径长度的指数形式。

【例 3-18】 对于图 3-2，从 $r123$ 到目标节点的花费是 0，因此

$cost_to_goal(r123) = 0$

继续从 $r123$ 使用最低花费优先搜索

$cost_to_goal(o123) = 4$

$cost_to_goal(o119) = 13$

$cost_to_goal(o109) = 29$

$cost_to_goal(b4) = 36$

$cost_to_goal(b2) = 39$

$cost_to_goal(o103) = 41$

$cost_to_goal(b3) = 43$

$cost_to_goal(b1) = 45$

到这个阶段向后搜索停止了。这里注意两点：首先，如果一个节点没有 $cost_to_goal$ 值，那么不存在从该节点到目标节点的路径；其次，Agent 可以很快地确定任意一个节点到目标节点的最低花费值路径的下一段弧。例如，如果 Agent 在 $o103$ ，决定到 $r123$ 的最低花费路径，需要比较 $4+43$ (通过 $b3$ 的花费)和 $12+29$ (直接到 $o109$ 的花费)，可以很快决定去 $o109$ 。

当建立 $cost_to_goal$ 函数时，搜索者会隐式地决定选择哪个邻居到达目标，它不是在运行时就确定哪个邻居在最优路径上，而是储存该信息。

104

动态规划搜索可以用来为 A^* 算法和分支界限算法建构启发信息。构建启发函数的一种方式简化问题(如去掉一些细节)，直到被简化的问题有一个足够小的状态空间。动态规划算法可以用来在简化了的问题中找到最优的路径。这个信息也可以用来作为原始问题的启发信息。

A^* 算法的最优性

对于一个搜索算法，如果不存在能使用更少的时间或空间或扩展更少的节点的其他算

法,同时保证解的质量,那么这个算法就是**最优的**(optimal)。最优搜索算法每次都会选择正确的节点。然而,因为我们不能直接实现它,所以这个算法的详细说明并不有效。这样一个算法是否能实现是一个开放问题(就像是否 $P=NP$)。然而,似乎有一个能被证明的陈述。

A^* 的最优性(optimality of A^*):在只使用弧花费和从起始节点到目标节点花费的启发式估计的搜索算法中,没有算法能比 A^* 算法扩展更少的节点并保证能找到最低花费路径。

证明概述:除非算法扩展了每一个路径 p ,这里 $f(p)$ 低于最优路径的花费,否则只给定弧线花费以及启发信息,我们不知道 p 是否是一条最低花费的路径。更正式地,假设一个算法 A' 找到了问题 P 的解,这里某些路径 p 没有扩展,这样 $f(p)$ 小于找到的解。假设有另一个问题 P' ,除了有一条花费为 $f(p)$ 的通过 p 的路径,其他与问题 P 一样。算法 A' 不能分辨 P' 和 P ,因为它不扩展路径 p ,所以它对问题 P' 会返回和 P 一样的解,但是对于 P 找到的解对于 P' 来说不是最优解,因为这个解比经过 p 的路径有更高的花费。因此,算法不能保证找到最低花费的路径,除非它找到所有的 f 值比最优路径低的路径,这样就必须找到 A^* 所找到的所有的路径。

反例:尽管这个证明似乎合理,但是有的算法扩展更少的节点。考虑一个算法,它使用像 A^* 算法的向前搜索以及向后的动态规划搜索方法,这两种方法在某些方面是交叉进行的(例如,向前和向后交叉进行)。向后的搜索会建立一个表: $cost_to_goal(n)$,指的是从 n 到目标节点的实际花费,它有边界值 b ,它已经扩展了所有到目标花费低于 b 的路径。向前搜索使用 $cost(p)+c(n)$ 上的优先级队列,这里 n 是路径 p 的最后一个节点,如果这个值已经算出 $c(n)$ 是 $cost_to_goal(n)$,否则 $c(n)$ 是 $\max(h(n), b)$ 。直觉上,如果从路径的最后一个节点 p 到目标节点存在路径,或者它使用向后搜索已经扩展的路径,或者使用花费至少为 b 的路径。这个算法保证能找到最低花费的路径,扩展比 A^* 更少的节点(见习题 3.11)。

总结:上面的反例似乎意味着 A^* 的最优性是错误的。然而,证明也有一定的吸引力,这不应被彻底忽视。 A^* 不是所有算法中最优的,但是对于所有向前搜索的方法,这个证明似乎是正确的(见习题 3.12)。

3.8 本章小结

- 许多实际问题可以抽象为在图中进行路径搜索的问题。
- 宽度优先搜索和深度优先搜索可以在没有任何图以外的额外信息的情况下通过图来找到路径。
- A^* 搜索可以使用启发函数,来估计从一个节点到目标节点的花费。如果这个估计低于实际花费, A^* 能保证找到最低花费路径。
- 迭代深化搜索和深度优先分支界限搜索可以用来找到最低花费路径,且能比像 A^* 之类的算法节省更多的内存,因为 A^* 算法会储存多条路径。
- 当图很小的时候,动态规划搜索可以用来纪录从每一个节点到目标节点的最低花费路径的实际花费,这可以用来找到最优路径的下一段弧。

3.9 参考文献及进一步阅读

在运筹学研究、计算机科学、人工智能等领域有很多有关搜索技术的文献。搜索早期被认为是人工智能基础。人工智能强调启发式搜索。

基础的搜索算法讨论可参考 Nilsson[1971]。Pearl[1984]中有详细的启发式搜索方法。Hart、Nilsson 和 Raphael[1968]中有 A* 算法的进一步探讨。

Korf[1985]中有关于深度优先迭代深化搜索的描述。

分支界限搜索方法在运筹学研究中有进一步探讨，在 Lawler 和 Wood[1966]中有描述。

动态规划算法是一种通用算法，在本书的其他部分作为双重搜索方法。这种特殊算法是 Dijkstra [1959]最先提出的。Cormen、Leiserson、Rivest 和 Stein[2001]有关于动态规划这类算法的更多详细介绍。

Culberson 和 Schaeffer[1998]首先提出使用动态规划算法作为 A* 搜索的启发信息这一思想。Felner、Korf 和 Hanan[2004]有更深入的介绍。

Minsky[1961]讨论了岛算法和问题简化思想。

105
}
106

3.10 习题

- 3.1 评论下面的话：人工智能的一个主要目标是为图搜索问题建立一般的启发信息。
- 3.2 在边界中的任一元素被选中这一方面来说，哪一种路径搜索程序是公平的？在没有环路的有限图，或者有环路的有限图，或者无限图(有有限的分支元素)中考虑这个问题。
- 3.3 看图 3-13 的在网格中寻找路径的问题，要找的是从 *s* 到 *g* 的路径。可以在水平方向和竖直方向上移动，一次只能一个方向。阴影部分表示禁止移动。
 - (a) 在图 3-13 的网格中，为从 *s* 到 *g* 的深度优先搜索的路径上各个扩展的节点标号。操作的顺序是上、左、右、下。假设有循环检查，
 - (b) 对于同样的网格，标号扩展的节点，是为了得到从 *s* 到 *g* 的最优先搜索解。曼哈顿距离应当作为评价函数。曼哈顿距离是两点的 *x* 轴方向上的距离加上 *y* 轴方向上的距离。它对应的是在网格中沿着城市旅游。假设有路径剪枝，那么发现的第一条路径是哪条？
 - (c) 对于同样的网格，标号扩展的节点，是为了得到从 *s* 到 *g* 的启发式深度优先搜索解。曼哈顿距离作为评价函数。假设存在路径循环检查，那么找到的路径是什么？
 - (d) 用 A* 搜索方法标号节点的顺序，使用多路径剪枝，找到的路径是什么？
 - (e) 描述用动态规划算法如何解决相同问题。给出每个节点的 *dist* 值，描述找到的路径。
 - (f) 按照经验，哪种搜索方法最适合这个问题？
 - (g) 假设图在各个方向上延伸。也就是说，这个图没有界限，但是 *s*、*g* 和那些阴影障碍物都在相同的地方。用哪种方法会找不到路径？哪种是最好的方法？为什么？
- 3.4 问题是使用图搜索来设计视频演示。假设存在一个视频片段的数据库，还有它的时间长度和主题，如下：

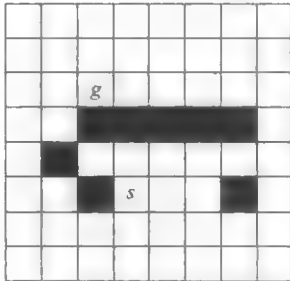


图 3-13 一个格搜索问题

片段	长度	包含的主题
seg0	10	[welcome]
seg1	30	[skiing, views]
seg2	50	[welcome, artificial_intelligence, robots]
seg3	40	[graphics, dragons]
seg4	50	[skiing, robots]

假设一个节点包含一对元素：

(*To_Cover*, *Segs*)

这里 *Segs* 是必须要表示出来的一个列表，*To_Cover* 也是必须表示的一系列主题。假设没有列表包含任何一个 *To_Cover*。

节点的邻居通过 *To_Cover* 进行选择。对于每一个列表有邻居包含已经选择的主题。(这部分作用是要考虑这些相邻点的确切结构。)

例如, 给出上述数据库, 节点 $\langle welcome, robots \rangle, [] \rangle$ 的邻居是 $\langle [], seg2 \rangle$ 和 $\langle robots, seg0 \rangle$, 假设选择了 *welcome*。

因此, 每一个弧线添加一段, 但是可以包含一个或多个主题。假设弧线的成本等于片段添加的时间。

目标是设计一个描述, 包含了 *MustCover* 所有的主题。起始节点是 $\langle MustCover, [] \rangle$, 目标节点是 $\langle [], Presentation \rangle$ 。从起始节点到目标节点的花费值是这个描述的时间。因此, 最优的描述是包含了 *MustCover* 所有的主题的最短的描述。

(a) 假设目标包含了所有主题 $[welcome, skiing, robots]$ 。假设算法总是选择每个节点最左边的主题为其查找邻居。画出最低花费优先算法所有扩展的节点直到找到最终的解。这就会显示所有扩展的节点, 到找到目标节点时, 边界就会出现。

(b) 给定一个非平凡的启发函数 h , 它是实际花费的低估值。(注意对于所有的节点 $h(n)=0$ 是平凡的启发函数。)它满足启发函数的单调性质吗?

3.5 画出两个不同的图, 标出初始节点和目标节点, 其中一个图表示向前搜索比向后搜索好, 另外一个图表示向后搜索比向前搜索好。

3.6 实现迭代深化 A^* 算法。基于图 3-10 中的迭代深化搜索方法。

3.7 假设我们要找一个从初始节点到目标节点的路径, 不是寻找最优路径, 而是寻找花费值比最低花费多 10% 的路径。建议迭代深化 A^* 算法的替代方法可以得到解。为什么这有利于迭代深化 A^* 搜索?

3.8 如何修改深度优先分支界限搜索方法, 来找到花费值比最低花费多 10% 的路径。这个算法与上面一题中的 A^* 算法的优化方案相比怎么样?

3.9 对于迭代深化搜索的分母 $b-1$, 当 $b \approx 1$ 时, 这就不不是一个很好的近似。给出一个当 $b=1$ 时更好的迭代深度复杂性的估计。这一章中其他算法的复杂度怎么算? 提示: 分支系数越接近 1, 迭代深化的分母越小。

3.10 双向搜索必须确保何时两个边界相交。为下面每个决定确定何时相交:

(a) 宽度优先搜索和深度界限深度优先搜索。

(b) 迭代深化搜索和深度界限深度优先搜索。

(c) A^* 算法和深度界限深度界限搜索。

(d) A^* 算法和 A^* 算法。

3.11 考虑前面 3.7.6 节中“ A^* 算法的最优性”处讨论的反例的算法, 回答:

(a) 什么时候算法停止? (提示: 到向前搜索找到目标, 算法才会停止。)

(b) 应该保留什么样的数据结构?

(c) 完整的描述算法。

(d) 描述怎么样找到最优路径。

(e) 给出一个例子说明它扩展了比 A^* 少(多)的节点。

3.12 表述 A^* 算法的最优性, 对于指定算法的种类, A^* 是最优的。给出证明。

3.13 图 3-11 中深度优先分支界限搜索方法类似于深度界限搜索。因为它们都是如果花费低于界限值时, 只找到一条解路径。对于一个特定的深度界限, 如果没有解, 它是怎样结合迭代深化搜索来增加深度界限的? 如果没有解, 在有限图中, 算法会返回一个 \perp 。算法应当允许界限任意增加, 然后当解存在时, 返回最优(最低花费)的路径。

特征和约束

每项任务都会有其约束的一面，要想毫无怨言地解决问题，需要依靠一些神奇的连接和链条来松弛僵化的大脑。虽然它们限定了条条框框，却又神奇地解放了思维。

——James Falen

根据特征来描述状态，然后基于这些特征进行推理，这种推理方式一般情况下会比用状态显式推理获得更好的效果。通常这些特征是非独立的，并且存在**硬约束**(hard constraint)，这些硬约束指定了变量赋值的合法组合。正如 Falen 在诗中强调，发现并利用这些约束去解决问题。在规划和调度问题中常常有这类范例，例如**智能体**(Agent)必须为每个确定执行的动作分配一个时间，在这些被执行的动作中则存在时间上的约束，并且这些约束说明了多个动作必须完成一个实际目标。除硬约束外，也存在**软约束**(soft constraint)，这些软约束可以指定一些具体的偏好值。本章将说明如何生成满足硬约束的赋值和如何最优化一个软约束集合。

4.1 特征和状态

任何实际问题都可能存在大量的状态，所以 Agent 不能根据其状态进行推理。而且，大多数问题不会配备一个明确的状态列表，这些状态往往是由一些隐含的**特征**(feature)所描述。当描述一个真实的状态空间时，通常是描述组成状态的特征，而不是直接枚举各个状态。

111

特征和状态的定义是交织关联的，因此可以通过其中一个对另一个进行描述性的定义。

- 根据特征，状态可以被定义为：特征作为基元，一个状态则对应了每一个特征的一种赋值。
- 根据状态，特征可以被定义为：状态作为基元，一个特征是状态构成的一个函数。对于给定的状态，函数返回该状态下的特征值。

每一个特征都有一个**域**(domain)，这个域是它可取值的集合。特征的域就是在全体状态下函数的值域。

【例 4-1】 在图 1-8 的电气环境中，每一个形容上、下的开关位置都有可能被看做是一个特征。每一个形容灯泡亮或灭的灯泡位置也有可能被看做是一个特征。每一个说明其正常工作还是中断的组件都可以被看做是一个特征。一个状态包含了每个开关的位置、每个设备的情况，等等。

如果将特征作为基元，一个状态就是为每个特征赋值。例如，一个状态可以被这样描述：开关 1 置上，开关 2 置下，保险丝 1 正常，导线 3 中断，等等。

如果将状态看成基元，函数可以举例为开关 1 位置。这个位置是一个状态函数，在某些状态下它处于置上，其余的状态下它处于置下。

根据特征进行推理的主要的优势在于节省了计算量。例如，一个**二元特征**(binary

feature)的域有两个取值,那么由少量的特征就能产生大量的状态:

- 10 个二元特征可以表示 $2^{10}=1\,024$ 个状态。
- 20 个二元特征可以表示 $2^{20}=1\,048\,576$ 个状态。
- 30 个二元特征可以表示 $2^{30}=1\,073\,741\,824$ 个状态。
- 100 个二元特征可以表示 $2^{100}=1\,267\,650\,600\,228\,229\,401\,496\,703\,205\,306$ 个状态。

根据 30 个特征进行推理要比根据超过 10 亿个状态推理简单得多。100 个特征不算多,但是要是用 2^{100} 个状态进行推理很明显是不可能的。很多问题都有数千或者数百万的特征(这种情况,用状态推理更是不可能的)。

通常特征是非独立的,因为在不同特征的取值上可能存在约束。考虑给定的特征和约束来确定可能的状态成为一个重要问题。

112

4.2 可能世界、变量和约束

为了保持形式上的简化和通用,在展开特征的概念中没有明确地考虑时间(约束)。本节将根据可能世界的概念来描述约束满足问题。

不更改建模的情况下,在变量和特征中存在一对一的直接对应,这种对应同样存在于状态和可能世界之间。

可能世界(possible world)是世界(真实世界或虚幻世界)存在的一种可能方式。例如,当表述一个填字游戏时,可能世界则对应了填字游戏被填写的方式。在电气环境中,可能世界则是指每个开关的位置(置上或置下)和每个设备的状态(正常或中断)。

代数变量(algebraic variable)经常被看作是表达可能世界特征的符号,因此可能世界可用代数变量进行表达。代数变量通常以大写字母开头,并且每个代数变量 V 都有一个关联的域,记做 $dom(V)$,这个域是变量可取值的集合。

本章中所提及的代数变量可以被简单地理解为变量(variable)。而代数变量和变量在逻辑上的区别将在第 12 章进行讨论。在第 6 章将说明代数变量和随机变量在概率理论中是相同的。

离散变量(discrete variable)的域是有限的或者是可数无限的。例如离散变量的一个实例是**二元变量**(boolean variable),它的域是 $\{true, false\}$ 。如果 X 是一个二元变量,书写时用 x 来等价表示它的大写体 $X=true$,用 \bar{x} 表示 $X=false$ 。当然对应离散变量也存在非离散变量,例如一个变量的域对应了实数线上的一个子集,则这个变量是**连续变量**(continuous variable)。

【例 4-2】 作为一个特定的类,课堂时间(Class_time)这一变量可以由起始时间来表达。那么课堂时间的域可用如下的一组时间的集合表达:

$$dom(Class_time)=\{8,9,10,11,12,1,2,3,4,5\}$$

Joe 的身高(Height_joe)这一变量指的是一个具体的人在一个具体时间的身高,而它的域则是一定身高范围内以厘米为单位的实数集合。如果在一个具体时间正在下雨,那么下雨(Raining)可以被看做是一个二元随机变量,此时这个变量取值为 true。

【例 4-3】 考虑图 1-8 中所描述的电气系统的域:

- S_1_pos 可以被认为表示开关 S_1 位置的离散二元变量,它的域是 $\{up, down\}$ 。当 $S_1_pos=up$ 时,表示开关 S_1 置上;当 $S_1_pos=down$ 时,表示开关 S_1 置下。
- S_1_st 可以被认为表示开关 S_1 状态的变量,它的域是 $\{ok, upside_down, short, intermittent, broken\}$ 。当 $S_1_st=ok$ 时,开关 S_1 正常工作;当 $S_1_st=upside_down$ 时,开关 S_1 倒置。

down 时, 开关 S_1 被倒置安装; 当 $S_1_st=short$ 时, 开关 S_1 短路并相当于一根导线; 当 $S_1_st=intermittent$ 时, 开关 S_1 间断性工作; 当 $S_1_st=broken$ 时, 开关 S_1 中断, 不允许电流通过。

- $Number_of_broken_switches$ 是一个整数值变量, 它代表着中断的开关数。
- $Current_w_1$ 是一个实值变量, 它代表着当前通过 w_1 导线的安培数。当 $Current_w_1=1.3$ 时, 表示当前有 1.3 安培通过 w_1 导线。在变量和常量中也允许不等式作为二元特征; 例如, $Current_w_1 \geq 1.3$ 为真时, 表达至少有 1.3 安培通过 w_1 导线。

113

符号和语义

代数变量是符号。

有别于其他符号, 在计算机内部**符号**(symbol)是一个二进制序列。一些符号有其固有的解释, 例如, 表达数字的符号和表达文字的符号。而在程序设计语言中一些符号没有固定的意义。在 Java 语言中, 从 Java 1.5 版本开始它们被称为枚举类型(enumeration type)。在 LISP 语言中, 指的是原子(atom)。通常情况下, 它们以索引的方式被应用在给出打印名字的符号表中。在这些符号中唯一的操作是判断两个符号是否相同。

对于计算机用户来说, 符号是有意义的。当用户输入约束或者解译输出结果的关联意义时, 需要借助组成约束和输出结果的符号。用户将符号和概念、目标联系在一起。例如, 对于计算机, 变量 $HarrysHeight$ 仅仅是一个二进制序列, $HarrysHeight$ 和 $SuesHeight$ 没有关联。对于一个人来说, 这可能是在具体时间下某人在具体单位上的身高。

变量-值这样一个配对的相关意义必须满足**明晰性原则**(clarity principle): 一个**全知智能体**(omniscient Agent)——即一个虚构的 Agent, 该 Agent 知道和所有符号关联的意义和真理——这个 Agent 可以决定每一个变量的值。例如, 如果这个具体的人被提及, 并且具体时间和单位被指定, 那么 $height\ of\ Harry$ 满足明晰性原则。例如, 以厘米为单位我们可以推理 J.K. 罗琳第二部书中起始部分 Harry Potter 的身高。这和以英寸为单位在同本书中尾声部分的 Harry Potter 的身高是有区别的(尽管它们之间肯定是有联系的)。如果想查阅在这两个不同时间的 Harry 的身高, 则必须有两个不同的变量。

当声明约束时, 必须有统一的含义, 也就是说, 对于相同的变量和相同值必须被赋予相同的含义, 并且可以用这个含义去解释输出。

这段文字最主要的内容是说明符号是有含义的, 因为赋予了它意义。对于本章来说, 假设计算机不知道符号的含义。一个计算机只有通过感知和操纵环境, 才能知道符号的含义。

114

【例 4-4】 填字游戏是一个经典的约束满足问题。根据变量填字游戏有如下两种不同的表达:

1) 一种表达是将能够表达字方向(横或纵)的方格进行标号, 将这些具有编号的方格作为变量, 变量的域则是一组可能被填写的字的集合。对于每个变量, 一个可能世界则对应了每个字的一种赋值。

2) 另一种表达是把每个单独的方格(非编号)当做变量, 每个变量的域是字母表中字母的集合。每个方格内填写一个字母对应了可能世界。

根据变量定义可能世界或者根据可能世界定义变量:

- 变量作为基元, **可能世界**则对应了为每个变量赋一个值的全赋值(total assignment)。

- 可能世界作为基元，变量是从可能世界到变量域的一个函数；对于给定的可能世界，函数的返回值是变量在可能世界中的变量值。

【例 4-5】 如果存在两个变量， A 的域是 $\{0, 1, 2\}$ ， B 的域是 $\{true, false\}$ ，那么存在 6 个可能世界，这里命名为 w_0, \dots, w_5 。对于变量和可能世界的可能赋值方式如下所示：

- w_0 : $A=0$ 且 $B=true$
- w_1 : $A=0$ 且 $B=false$
- w_2 : $A=1$ 且 $B=true$
- w_3 : $A=1$ 且 $B=false$
- w_4 : $A=2$ 且 $B=true$
- w_5 : $A=2$ 且 $B=false$

【例 4-6】 在制定一组游客的旅行计划时，一个交易 Agent 可能被要求去安排一组活动集合。对于每个活动则需要两个变量：一个是日期的；一个是地点的。日期的域是进行活动的可能时间集合；地点的域是活动发生的可能城镇集合。一个可能世界则对应了每个活动发生的时间和地点的一种赋值。

4.2.1 约束

在很多领域，并不是变量的所有可能赋值都被允许。一个硬约束(hard constraint)或者简单约束指定了变量的合法赋值组合。

作用域(scope)或者**方案(scheme)**是一组变量的集合。在作用域 S 上的**元组(tuple)**是对 S 中每个变量的一种赋值。在作用域 S 上的一个**约束(constraint)** c 是 S 上一个元组的集合。约束**涉及(involve)**在其作用域内的每一个变量。

如果 S' 是变量集合，且 $S \subseteq S'$ ， t 是 S' 上的一个元组。如果受限于变量 S 的 t 在 c 中，那么称 t **满足(satisfy)**约束 c 。

115

约束可以用关系数据库中的术语来定义。约束和关系数据库的最大区别在于约束指定了合法值，而数据库关系是指定了在某些情况下什么为真。相对于在一个表中外延地(extensionally)表达每个赋值，约束通常是依据谓词(二元函数)被**内涵地(intensionally)**定义。具体的定义既可以用合法赋值表示，也可以用非法赋值表示。

【例 4-7】 考虑在可能时间上三个活动的一个约束。 A 、 B 、 C 是三个变量代表该时间下的每个活动。假设每个变量的域为 $\{1, 2, 3, 4\}$ 。

约束可以用表格的形式被写出，并给出如下的合法组合：

A	B	C
2	2	4
1	1	4
1	2	3
1	2	4

其中 $\{A, B, C\}$ 是作用域。每一行是一个元组，该元组是在作用域限制下的每个变量的合法赋值。第一个元组是

$\{A=2, B=2, C=4\}$

赋予 A 值为 2， B 值为 2， C 值为 4。这个元组指明了变量 4 种合法赋值中的一种。

用上述表格的约束可以给出这样的元组 $\{A=1, B=2, C=3, D=3, E=1\}$ ，因为该元组对作用域变量的赋值均为合法赋值。

这种约束性的合法赋值也可以用一阶谓词逻辑公式更内涵地描述，那么上述约束可被表达为如下：

$$(A \leq B) \wedge (B < 3) \wedge (B < C) \wedge \neg (A = B \wedge C = 3)$$

这里 \wedge 是逻辑与， \neg 是逻辑非。这个公式表达了这样的约束： A 小于等于 B ， B 小于3且 B 小于 C ，并且不存在 A 等于 B 且 C 等于3的情况。

一元约束(unary constraint)是作用在单个变量上的约束(例如： $X \neq 4$)。**二元约束**(binary constraint)是作用在一对变量上的约束(例如： $X \neq Y$)。通常， k 元约束有一个大小为 k 的作用域。

对于每个约束，如果可能世界 w 中变量的赋值都满足在约束条件下作用域内变量的相应约束，那么就说可能世界 w 满足约束集。这种情况下我们说可能世界是约束的一个模型(model)。换言之，一个模型就是满足所有约束的一个可能世界。

116

【例 4-8】 假设配送机器人必须完成一组配送活动 a 、 b 、 c 、 d 、 e 。假设每个活动发生的次数可以是1、2、3、4中的任意次数。变量 A 表示活动 a 发生的次数，类似定义其余活动的发生次数。代表每个配送可能发生次数的变量域如下：

$$\text{dom}(A) = \{1, 2, 3, 4\}, \text{dom}(B) = \{1, 2, 3, 4\}, \text{dom}(C) = \{1, 2, 3, 4\}$$

$$\text{dom}(D) = \{1, 2, 3, 4\}, \text{dom}(E) = \{1, 2, 3, 4\}$$

假设如下约束是必须被满足的：

$$\{(B \neq 3), (C \neq 2), (A \neq B), (B \neq C), (C < D), (A = D), (E < A), (E < B), (E < C), (E < D), (B \neq D)\}$$

这就是寻找到一个模型，使得每个变量的赋值都能满足所有约束条件。

【例 4-9】 考虑例 4-4 中关于填字游戏约束的两种表达。

1) 对于填字游戏来说，如果它的域是许多单词，那么约束就是一对单词相交叉的字母必须相同。

2) 如果它的域是英文字母，那么约束则是字母的连续序列必须构成一个合法的单词。

【例 4-10】 例 4-6 中，对于那些典型的约束也有很多种描述方式。可能某些具体的活动需要安排在不同的日期，也可能另外一些活动需要安排在同一城镇的同一天；可能是某些活动必须被安排在其他活动之前，也可能两个活动之间必须相隔一定的天数，还有可能是三个活动不能被安排在连续的三天。

4.2.2 约束满足问题

一个约束满足问题(constraint satisfaction problem, CSP)由以下三者构成：

- 一个变量集合；
- 每个变量都有一个域；
- 一个约束集合。

其目的是选择出使可能世界满足约束的每个变量值，因此需要约束的模型。

一个有限的约束满足问题(CSP)要求变量集是有限的，并且变量的域是有限的。尽管针对无限的域甚至连续域的 CSP 设计了一些方法，但是本章的许多方法仅针对有限 CSP 有效。

117

当每个变量被看做是一个单独的维度，这类 CSP 问题就转化为多维的角度。多维使 CSP 更难以处理，却也提供了可利用的结构。

对于给定的 CSP，可以进行以下的处理：

- 判断是否存在一个模型。
- 寻找一个模型。
- 寻找全部的模型或者枚举模型。
- 计算模型的数量。
- 对于给定评估模型的度量标准，寻找一个最优的模型，见 4.10 节。
- 判断一些表达是否适用于所有模型。

本章主要研究找到一个模型的问题。有一些方法也可以判断是否无解。比较特殊的是，有一些方法在有解的情况下可以给出一个模型，但是无解的情况下却不能告之无模型存在。

约束满足问题是广泛存在的，因此寻找有效的方法解决这类问题是具有现实意义的。对一个有限域的 CSP 问题，判断它是否存在一个模型是 NP 难的（详见 5.2.2 节中“不确定性选择”的内容），且在最坏的情况下，不存在一个已知算法可以在非指数时间内解决这类问题。但是，不能因为一个问题是 NP 难的，就断言所有实例都是难以解决的。一些实例提供了可利用的结构。

4.3 生成-测试算法

任何有限的 CSP 问题都可以用穷举的生成-测试算法来解决。赋值空间 (assignment space) D 表示所有变量的赋值集合，它对应了所有可能世界组成的集合。 D 中的每个元素都代表对所有变量的一种赋值。算法返回值是符合所有约束的赋值。

因此，生成-测试 (generate-and-test) 算法如下：依次检查每个全赋值，如果一种全赋值满足了所有约束，那么返回这种赋值。

【例 4-11】 例 4-8 中赋值空间是

$$D = \{ \{A=1, B=1, C=1, D=1, E=1\}, \{A=1, B=1, C=1, D=1, E=2\}, \dots, \\ \{A=4, B=4, C=4, D=4, E=4\} \}$$

本例中需要测试 $|D| = 4^5 = 1\,024$ 个不同的赋值。在习题 4.1 的填字游戏的例子中，有 $40^6 = 4\,096\,000\,000$ 个可能的赋值。

如果 n 个变量中每个变量的域大小为 d ，那么 D 中有 d^n 个元素。如果存在 e 个约束，那么测试的约束实例总数为 $O(ed^n)$ 。当 n 很大时，这个问题就会变得难以解决，因此我们需要找到一种有效的求解方法。

118

4.4 使用搜索求解 CSP

生成-测试算法在检查是否满足约束之前需要为所有变量赋值。因为单独的约束仅仅包含一个变量子集，所以在所有变量都被赋值之前，已经有一部分约束能被测试。如果存在部分赋值和某约束不一致，那么包含这部分赋值的任意完整赋值都是与约束不一致的。

【例 4-12】 在例 4-8 配送计划问题中， $A=1$ 且 $B=1$ 的赋值方式与约束 $A \neq B$ 是不一致的。如果变量 A 和 B 被首先赋值，那么这种不一致就会在为 C 、 D 、 E 赋值之前被发现，这样可以节省大量的工作。

一个替代生成-测试算法的方式是构建一个搜索空间，之前章节所提供的搜索策略在

这个搜索空间内可以被使用。搜索问题可以如下定义：

- 将一些变量子集的赋值作为节点。
- 在节点 N 中选择一个未被赋值的变量 V ，在含有 V 的每个赋值中选取不违反任何约束的赋值作为 N 的邻居节点。

假设节点 N 表示赋值 $X_1=v_1, \dots, X_k=v_k$ 。为了找到 N 的邻居，选择变量 Y ，该变量不在集合 $\{X_1, \dots, X_k\}$ 中。对于每一个值 $y_i \in \text{dom}(Y)$ ，如果 $X_1=v_1, \dots, X_k=v_k, Y=y_i$ 符合约束条件，那么 $X_1=v_1, \dots, X_k=v_k, Y=y_i$ 为 N 的邻居节点。

- 开始节点为空节点，表示没有任何变量被赋值。
- 一个目标节点则是对每个变量都进行赋值的节点。这种节点只有在赋值满足所有约束的条件下才存在。

这种情况下，寻找的不是兴趣路径，而是目标节点。

【例 4-13】 假设存在一个含有 A 、 B 、 C 三个变量的 CSP 问题，且每个变量的域为 $\{1, 2, 3, 4\}$ 。假设约束为 $A < B$ 且 $B < C$ 。那么图 4-1 可能是一个满足这种约束的搜索树。图中一个节点对应了从根节点到该节点的所有赋值。不符合约束的潜在节点将被剪枝掉，用 \times 来标记。

最左边的 \times 代表赋值 $A=1, B=1$ ，这不符合 $A < B$ 的约束，因此被删除掉。

这个 CSP 问题有 4 个解。最左边的解是 $A=1, B=2, C=3$ 。搜索树的大小和算法的效率取决于每次选取的变量。如果使用先 A 再 B 最后 C 的静态分割顺序，那么效果将没有本例中使用的动态顺序好。尽管变量顺序不同，但结果是相同的。

在本例中，使用生成-测试算法需要测试 $4^3 = 64$ 个赋值方式。但用搜索的方法，只生成 22 个赋值方式。

119

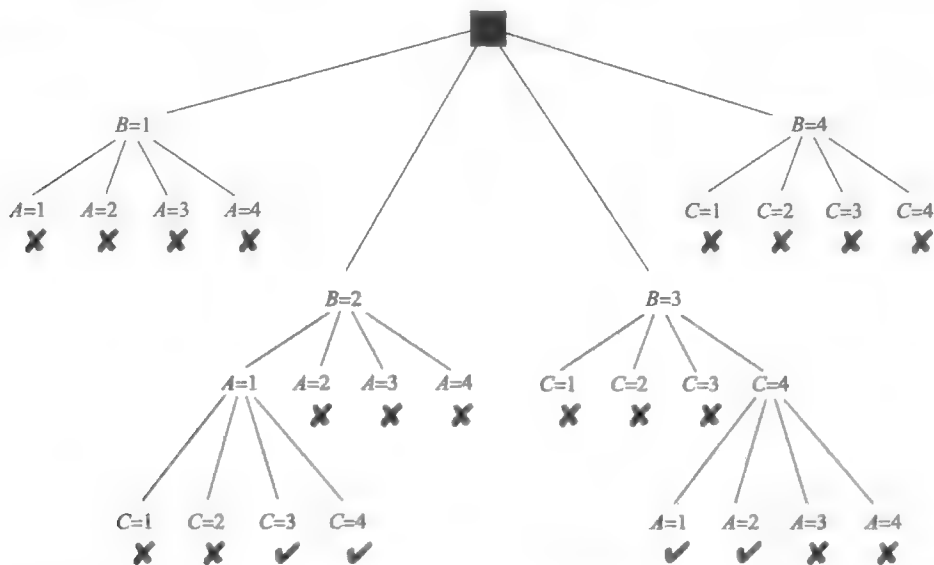


图 4-1 例 4-13 中 CSP 的搜索树

使用深度优先的搜索方法，通常称为回溯(backtracking)。该方法比生成-测试算法更为有效。生成-测试算法相当于在到达叶子节点之前都不检查是否满足约束条件。早检查约束条件就可以修剪子树，进而减少了不必要的搜索。

4.5 一致性算法

尽管深度优先搜索算法在赋值的搜索空间内比生成-测试算法有了实质性的提高,但是该方法仍有多个缺陷可以被改进。

【例 4-14】 例 4-13 中, 变量 A 和 B 通过约束 $A < B$ 被联系起来。因为 $dom(B) = \{1, 2, 3, 4\}$, 所以 $A=4$ 与每一个可能的赋值组合后都不满足约束。而在图 4-1 回溯法中, B 与 C 的赋值过程中, 这个事实就被忽略。假如使用一个简单的手段将 4 这个值从 $dom(A)$ 中删除, 那么这种失效性就可以一劳永逸地避免。

对于由约束满足问题形成的整个约束网络的操作来说, 众多方法中一致性算法的思想是最好的:

- 每个变量为一个节点, 变量节点用圆形画出。
- 每个约束为一个节点, 约束节点用长方形画出。
- 对于每个变量 X , 与其相关的 D_X 是变量可能取值的一个集合。这个集合的初始值为变量的域。
- 对于每个约束 c , c 与它作用域内的每一个变量 X 都用弧连接表示 $\langle X, c \rangle$ 。

这样一个网络就称之为一个约束网络(constraint network)。

【例 4-15】 例 4-13 中, 有 A 、 B 、 C 三个变量, 且每个变量的域为 $\{1, 2, 3, 4\}$ 。约束为 $A < B$ 且 $B < C$ 。在图 4-2 的约束网络中, 有 4 个弧:

$\langle A, A < B \rangle$

$\langle B, A < B \rangle$

$\langle B, B < C \rangle$

$\langle C, B < C \rangle$



图 4-2 例 4-15 中 CSP 的约束网络

【例 4-16】 约束 $X \neq 4$ 有 1 个弧:

$\langle X, X \neq 4 \rangle$

约束 $X + Y = Z$ 有 3 个弧:

$\langle X, X + Y = Z \rangle$

$\langle Y, X + Y = Z \rangle$

$\langle Z, X + Y = Z \rangle$

最简单的情况是, 约束的作用域内仅有一个变量。这种情况下, 如果单变量的每个变量值都满足约束, 那么这个弧就是域一致(domain consistent)的。

【例 4-17】 约束 $B \neq 3$ 的作用域是 $\{B\}$ 。在该约束下, 且 $D_B = \{1, 2, 3, 4\}$ 时, 因为 $B=3$ 不符合约束, 因此弧 $\langle B, B \neq 3 \rangle$ 不是域一致的。假如 3 从 B 的域中删除, 那么弧将是域一致的。

假设约束 c 的作用域是 $\{X, Y_1, \dots, Y_k\}$ 。如果对于每个值 $x \in D_X$, 有 y_1, \dots, y_k , 其中 $y_i \in D_{Y_i}$, 使得 $c(X=x, Y_1=y_1, \dots, Y_k=y_k)$ 被满足。那么弧 $\langle X, c \rangle$ 是弧一致(arc consistent)的。

【例 4-18】 例 4-15 的网络中, 没有弧是弧一致的。因为 $A=4$ 时 B 中没有与之对应且

满足 $A < B$ 的值, 所以第一个弧不是弧一致的。如果把 4 从 A 的域内删除, 那么该弧为弧一致的。第二个弧同样不是弧一致的, 因为在 $B=1$ 时, A 中没有与之对应的值。

121

如果弧 $\langle X, c \rangle$ 不是弧一致, 那么 X 中则存在一些值, 这些值不能保证 Y_1, \dots, Y_k 与之对应使得约束成立。这种情况下, 为了保证弧 $\langle X, c \rangle$ 是一致的, 则需要判断变量的 D_X 中有哪些值是与其他变量值无法对应的, 将这些值从 D_X 中全部删除。

图 4-3 中给出了广义弧一致性 (generalized arc consistency, GAC) 算法。通过对 TDA (弧集) 集合中潜在的非一致弧和待处理的弧进行操作, 构建整个网络的弧一致。 TDA 初始时包含了图中所有的弧。当弧集不为空时, 从集合内选择删除的弧 $\langle X, c \rangle$ 需要被进一步分析。如果弧是非一致的, 那么修剪变量 X 的域使之变为一致。作为变量 X 修剪结果, 之前所有一致的弧都可能变成非一致的, 因此需要选择一些弧将其放回 TDA 。选择弧 $\langle Z, c' \rangle$, c' 是关于变量 X 的不同于 c 的约束, 且 Z 是与 c' 相关但是不同于 X 的变量。

```

1: procedure GAC( $V, dom, C$ )
2:   Inputs
3:      $V$ : 变量集合
4:      $dom$ : 函数, 函数  $dom(X)$  为变量  $X$  的域
5:      $C$ : 被满足的约束集合
6:   Output
7:     每个变量的弧一致域
8:   Local
9:      $D_X$  是每个变量  $X$  的值的集合
10:     $TDA$  是弧集合
11:    for each 变量  $X$  do
12:       $D_X \leftarrow dom(X)$ 
13:       $TDA \leftarrow \{ \langle X, c \rangle \mid c \in C \text{ 且 } X \in scope(c) \}$ 
14:      while  $TDA \neq \{ \}$  do
15:        select  $\langle X, c \rangle \in TDA$ ;
16:         $TDA \leftarrow TDA \setminus \{ \langle X, c \rangle \}$ 
17:         $ND_X \leftarrow \{ x \mid x \in D_X \text{ 并且一些 } \{ X=x, Y_1=y_1, \dots, Y_k=y_k \} \in c \}$ 
           对于所有  $i, y_i \in D_{Y_i}$ 
18:        if  $ND_X \neq D_X$  then
19:           $TDA \leftarrow TDA \cup \{ \langle Z, c' \rangle \mid X \in scope(c'), c' \text{ 不是 } c, Z \in scope(c') \setminus \{ X \} \}$ 
20:           $D_X \leftarrow ND_X$ 
21:      return  $\{ D_X \mid X \text{ 是一个变量} \}$ 

```

图 4-3 广义弧一致算法

【例 4-19】用 GAC 算法操作例 4-15 的网络。初始, 所有的弧都在 TDA 集合中。下面是一个选择弧的可能顺序:

- 假设算法首先选择了弧 $\langle A, A < B \rangle$ 。对于 $A=4$, B 没有与之对应可以满足约束 $A < B$ 的值。因此, 4 从 A 的域中被删除。因为在 TDA 外没有其他的弧, 所以本次移动操作没有弧被添加到 TDA 中 (且弧 $\langle A, A < B \rangle$ 从 TDA 中删除——译者注)。
- 假设接下来 $\langle B, A < B \rangle$ 被选中, 那么 1 将被从 B 的域内删除, 仍然没有元素被添加到 TDA 中 (且弧 $\langle B, A < B \rangle$ 从 TDA 中删除)。
- 假设 $\langle B, B < C \rangle$ 又被选中, 那么 4 将被从 B 的域内删除。因为 B 的域因另一个约束被缩减, 因此 A 也存在被约简的潜在可能, 所以必须将弧 $\langle A, A < B \rangle$ 重新添加到 TDA 中。

122

- 如果 $\langle A, A < B \rangle$ 成为本次选中的弧, 那么 3 将从 A 的域内删除 (且弧 $\langle A, A < B \rangle$ 从 TDA 中删除)。
- 在 TDA 中还有弧 $\langle C, B < C \rangle$ 。那么值 1 和 2 将从 C 的域内被删除。没有弧被添加到 TDA 中, 且 TDA 为空。

算法终止, 并生成 $D_A = \{1, 2\}$, $D_B = \{2, 3\}$, $D_C = \{3, 4\}$ 。虽然该方法没有完全解决问题, 但是却将问题大大简化了。

【例 4-20】 用 GAC 算法来解决例 4-8 的调度问题。图 4-4 的网络图已经构造了域一致 (将值 3 从 B 的域内删除, 值 2 从 C 的域内删除)。

假设算法首先选择了弧 $\langle D, C < D \rangle$ 。因为 $D=1$ 和 D_C 中的任何值都不一致, 这个弧就不是弧一致, 将 1 从 D_D 中删除 (弧 $\langle D, C < D \rangle$ 从 TDA 中删除)。 $D_D = \{2, 3, 4\}$, 且关于 D 的弧 $\langle A, A = D \rangle$, $\langle B, B \neq D \rangle$, $\langle E, E < D \rangle$ 被添加到 TDA 中, 其实这些弧已经在 TDA 中了。

假设接下来选中 $\langle C, E < C \rangle$, 那么 D_C 被约简为 $\{3, 4\}$, 且 $\langle D, C < D \rangle$ 被重新添加到 TDA 中。

假设选中 $\langle D, C < D \rangle$, 那么 D_D 被进一步约简为单一的 $\{4\}$ 。处理弧 $\langle C, C < D \rangle$ 后 D_C 为 $\{3\}$ 。处理弧 $\langle A, A = D \rangle$ 后 D_A 为 $\{4\}$ 。处理弧 $\langle B, B \neq D \rangle$ 后 D_B 为 $\{1, 2\}$ 。处理弧 $\langle B, E < B \rangle$ 后 D_B 为 $\{2\}$ 。最后, 处理弧 $\langle E, E < B \rangle$ 后 D_E 为 1。队列中剩余所有的弧都是一致的, 因此算法结束且 TDA 为空。约简后的变量域被返回。在本例中, 所有返回的域大小都为 1 且是唯一的解: $A=4, B=2, C=3, D=4, E=1$ 。

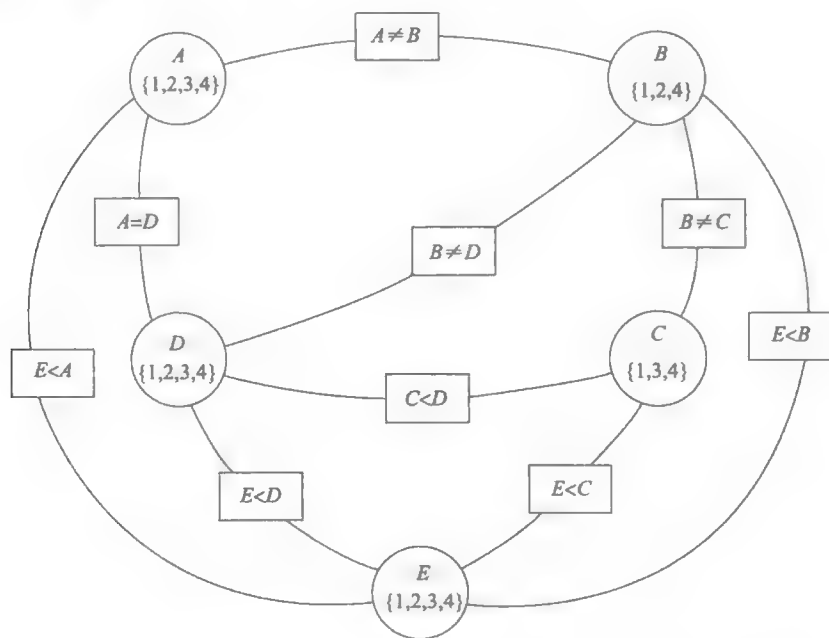


图 4-4 域一致约束网络。变量及其域用圆圈表示, 约束用长方形表示, 将变量和其相关的约束用弧线连接

GAC 算法不受选择弧的顺序影响, 终止时产生的结果是相同的, 换言之, 产生了相同的弧一致和相同的约简域。基于网络的终止状态, 以下三种情况有可能发生:

- 第一种情况, 某一变量的域为空, 这表明该 CSP 问题无解。注意, 只要某一域变为空, 那么在算法终止之前就可以把与这个变量节点相连接的节点的域都置为空。
- 第二种情况, 每个域都有一个单独的值, 这表明该 CSP 问题有且仅有一个解, 比如例 4-20。
- 第三种情况, 每个域都不为空, 并且至少有一个域内含有多个值。这种情况下, 则不知道是否存在一个解和解是什么。因此需要其他的方法来解决这个问题, 这类方法在后续章节中介绍。

123

下例将说明一个网络即使无解, 也有可能存在弧一致现象。

【例 4-21】 假设存在三个变量, 分别是 A 、 B 和 C , 每个变量的域为 $\{1, 2, 3\}$ 。存在这样的约束 $A=B$, $B=C$ 且 $A \neq C$ 。这种情况下, 使用任何单一约束都不能将域约简, 因此该例为弧一致。尽管存在弧一致, 但是却没有解。对于三个变量任何一种赋值方式都不能满足上述约束。

如果每个变量域的大小是 d 且有 e 个约束需要被测试, 那么 GAC 算法将在 $O(ed^3)$ 的时间内进行一致性检查。对于一些 CSP 问题, 例如约束图形是一个树形结构, 该方法可以基于变量数目以线性时间独自解决 CSP 问题。

弧一致的各种扩展技术也被研究。当域不是有限域时, 这些非有限的域可以用图形来描述, 而不是将它们的值罗列出来。当约束被扩展地表达时, 对约束的修剪也是可操作的: 如果一个值因变量 X 被删除, 那么所有涉及变量 X 的约束中这个值都可以被删除。高阶一致性技术, 例如 **路径一致** (path consistency), 可以同时分析多个变量的 K 元组, 而不仅仅是被一个约束连接的成对变量。例如在例 4-21 中, 同时考虑三个变量, 那么可以分析出此问题无解。后续介绍的增强型的弧一致方法在处理某些问题方面往往比高阶方法更有效。

124

4.6 域分割

另一个简化网络的方法是 **域分割** (domain splitting) 或者 **实例分析** (case analysis)。该方法的主要思路是将一个问题分割成许多不相交的实例, 然后分别解决每个实例。那么原始问题的所有解的集合就是每个实例的解的并集。

最简单的情况, 假设一个二元变量 X 的域是 $\{t, f\}$ 。所有的解要么是 $X=t$ 要么是 $X=f$ 。那么一个找到解的方法是, 首先设置 $X=t$ 找到所有 t 值下的解, 然后设置 $X=f$ 找到所有 f 值下的解。将一个变量赋值后给出一个更小的约简问题, 进而解决这个简化后问题。

如果一个变量的域有超过两个元素, 例如变量 A 的域是 $\{1, 2, 3, 4\}$, 那么有以下的方法去分割:

- 将域的每个值分割为一个实例。例如, 将 A 分割为 4 个实例 $A=1$, $A=2$, $A=3$ 和 $A=4$ 。
- 经常将域分割为两个不相交的子集。例如, 将 A 分割为两个实例, 一个实例是 $A \in \{1, 2\}$, 另一个实例是 $A \in \{3, 4\}$ 。

第一种单个分割的方式会产生更多的处理过程, 而第二种方式则允许以较少的步骤进行更多的修剪。例如, 不管 A 的值是 1 还是 2, 对于变量 B 都有相同的值可以被修剪, 那么第二种分割方式只需对这种情况分析一次, 并且不会对 A 中的每个元素再重新检查。节省操作的程度取决于怎样划分对域。

用域分割方法循环地解决实例、识别何时无解等价于 4.4 节中的搜索算法。在搜索时交叉使用弧一致算法会使搜索变得更加有效。

一个有效解决 CSP 问题的方法是在域分割操作之前使用弧一致算法对网络进行简化。也就是说，进行如下步骤来解决问题：

- 用弧一致方法来简化网络；
- 如果问题没有被解决，选择一个含多个元素的域进行分割，且循环解决每个实例。

该方法需要注意的是，在域分割之后不需要再从头开始弧一致操作。如果将变量 X 进行域分割，作为分割结果有些弧可能不再满足弧一致，TDA 可以用以下弧启动，且这些弧必须出自 $\langle Y, r \rangle$ ，其中 X 出现在 r 约束中， Y 是除 X 外出现在 r 约束中的变量。

【例 4-22】例 4-19 中，弧一致的方法虽然简化了网络，但是却没有解决该问题。在弧一致操作后，变量域内仍然含有多个元素。假设变量 B 被分割。有两个实例：

- $B=2$ 。这种实例下 $A=2$ 被删除。然后对 C 进行分割，得到两个解。
- $B=3$ 。这种实例下 $C=3$ 被删除。然后对 A 进行分割，得到两个解。

该方法的搜索树与图 4-1 的搜索树进行对比，弧一致的搜索空间更小，且对变量选择顺序不敏感。（图 4-1 中如果选用其他的变量选择顺序，搜索空间会更大。）

另一种增强域分割效果的手段如下：如果对变量的赋值拆分了图，每个被拆分的子部分可以被单独解决。那么该问题的完全解将是每个子部分的解的交叉乘积。这种方法可以减少很多计算，比如查询解的个数的计算或者寻找所有解的计算。例如，一个子部存在 100 个解，另一部存在 20 个解，那么问题将有 2000 个解。这比单独去寻找 2000 个解中的每一个解更有效。

利用命题结构

人工智能中一个基本的思想就是利用域中的结构。利用变量和约束的限制类可对 CSP 问题提供一种结构形式。这样的类别可以称之为命题可满足性 (propositional satisfiability) 问题的类别。这类问题具有以下性质：

- 二元变量：一个二元变量就是变量的域有 $\{true, false\}$ 两个值。给定一个二元变量 $Happy$ ，那么命题 $happy$ 表明 $Happy=true$ ，且 \overline{happy} 表明 $Happy=false$ 。
- 子句约束：一个子句 (clause) 形式为 $l_1 \vee l_2 \vee \cdots \vee l_k$ 的一种表达，这里每个 l_i 是一个文字。一个文字 (literal) 为二元变量的一个赋值。当可能世界中的一个子句被满足，或者说为真时，当且仅当组成子句的文字中至少有一个为真。

例如，子句 $happy \vee sad \vee \overline{living}$ 是在变量 $Happy$ 、 Sad 和 $Living$ 间的一个约束。当 $Happy$ 的值为真，或者 Sad 的值为真，或者 $Living$ 的值为假时，子句的值就为真。

一个子句是在一组二元变量上的约束，这组变量需要考虑消除一种赋值情况，就是使得全体文字为假的赋值。因此，子句 $happy \vee sad \vee \overline{living}$ 指明赋值 \overline{happy} 、 \overline{sad} 、 $living$ 是不被允许的。

当只有两个值的时候，从域中删除一个值等价于为变量赋了一个相反的值。例如，如果 X 的域是 $\{true, false\}$ ，那么从域中删除 $true$ ，等同于将变量 X 赋值为 $false$ 。

弧一致方法可以被用来修剪变量集合和约束集合。对一个二元变量赋值可以简化约束集合：

- 如果变量 X 赋值为 $true$ ，所有包含 $X=true$ 的子句都变为冗余，它们自动地被满足。那么这些子句就可以被删除。相似地，当 X 赋值为 $false$ 时所有包含 $X=false$ 的子句都可以被删除。

- 如果变量 X 赋值为 $true$ ，所有包含 $X=false$ 的子句都可以把 $X=false$ 从子句中删除。相似地，当 X 赋值为 $false$ 时，所有包含 $X=true$ 的子句都可以将 $X=true$ 从子句中删除。该步骤可以叫做单元归结(unit resolution)。

在一些修剪子句的步骤之后，子句可能存在仅包含一种赋值的情况，例如 $Y=v$ 。这种情况下，变量 Y 域内的其他的值就都可以被删除。对于赋值 $Y=v$ ，上述修剪操作可以被重复进行。如果所有的赋值都被从子句中删除，那么说明不满足约束。

【例 4-23】 考虑子句 $\bar{x} \vee y \vee \bar{z}$ 。当 X 被赋值为 $true$ 时，子句可以被简化为 $y \vee \bar{z}$ 。如果接下来 Y 被赋值为 $false$ ，子句可以被简化为 \bar{z} 。因此， $true$ 可以从 Z 的域内被删除。

相反地，如果 X 被赋值为 $false$ ，上述子句就可以被删除。

如果一个变量在所有剩余的子句中都用相同的值，且算法只需要发现一个模型，那么该变量可以被赋为这个相同的值。例如，如果变量 Y 仅以 $Y=true$ 的形式出现(如， \bar{y} 不是任何子句)，那么变量 Y 可以被赋值为 $true$ 。因为在设置 $Y=true$ 后，剩余的子句集合将是 Y 被赋予 $false$ 的子句集合的子集，因此这种赋值并不是删除所有的模型，而是将问题简化。在所有子句中仅含有一个值的变量被称之为纯符号(pure symbol)。

事实证明将修剪域和约束、域分割、纯符号赋值和有效地管理约束(即，判断何时除一个析取外其余所有的析取为假)应用到命题可满足问题是非常有效的手段。该算法以作者名字命名，简称为 DPLL。

4.7 变量消除

弧一致方法通过删除变量值简化了网络。一个更完整的思想是变量消除(Variable Elimination, VE)，该方法则通过删除变量来简化网络。

VE 的思想是逐个地删除变量。当删除一个变量 X 时，VE 将在剩余的一些变量中构建新的约束，新约束反映了 X 在所有其他变量上的影响。新约束替换了所有涉及变量 X 的约束并且形成了一个不含 X 的约简网络。由于构建了新约束，因此约简后的 CSP 的任何解都可被扩展为原有包含变量 X 的较大 CSP 的某一个解。所以除了创造新约束外，VE 提供一种构造解的方法，该方法从约简的 CSP 中选择一个解来构建包含变量 X 的 CSP 问题的解。

127

下述算法的描述将使用关系代数计算中的连接和投影操作。

消除变量 X 时， X 的影响力通过包含 X 的约束作用到其余变量。首先，算法收集所有涉及 X 的约束。让所有的这些关系进行连接操作形成关系 $r_X(X, \bar{Y})$ ，这里 \bar{Y} 是作用域 r_X 内的其他变量。然后把 r_X 投影到 \bar{Y} ，这个关系可以代替所有包含 X 的关系。那么这样就得到了一个包含更少变量的约简 CSP 问题，且约简后的 CSP 问题可以以此方法递归解决。一旦对于约简后的 CSP 问题生成了一个解，那么通过将解与 r_X 进行连接操作，就可以扩展这个解得到原始 CSP 问题的一个解。

仅剩下一个变量时，算法返回变量的域元素，因为这些元素同该变量的约束是一致的。

【例 4-24】 考虑包含变量 A 、 B 、 C 的 CSP 问题，这里每个变量的域为 $\{1, 2, 3, 4\}$ 。假设包含 B 的约束是 $A < B$ 和 $B < C$ 。可能存在其他的变量，但是这些其他变量与 B 不存在公共的约束，即使消除 B 也不能对这些变量添加任何新的约束。为了删除 B ，首先把包含 B 的关系进行连接操作：

A	B		B	C		A	B	C
1	2	\bowtie	1	2	$=$	1	2	3
1	3		1	3		1	2	4
1	4		1	4		1	3	4
2	3		2	3		2	3	4
2	4		2	4				
3	4		3	4				

由 B 得到 A 和 C 的关系, 将这个连接操作的结果投影到 A 和 C , 得到

A	C
1	3
1	4
2	4

这个 A 和 C 的关系包含了所有有关 B 的约束的信息。这个关系影响了剩余网络的解。

关于 B 的初始约束被 A 和 C 的新约束取代。因为不含变量 B , 所以解剩下的网络就会更简单。谨记得约简网络的一个解后, 然后通过连接关系构造一个包含变量 B 的解。

图 4-5 给出了消除变量的递归算法 VE_CSP , 该算法可以找到一个 CSP 问题的所有解。

递归的终止条件是仅有一个变量剩余的时候。这种情况下(第 8 行), 当且仅当在最后的连接关系中有“行”存在时才有解存在。注意这些关系是所有关系在一个单独变量上的体现, 因此这些关系是这个变量合法值的集合。这些关系的连接就是这些集合的交集。

在非终止条件下, 变量 X 被选择并消除(第 10 行)。变量的选择不影响算法的正确性, 但却影响算法的效率。为了消除变量 X , 算法将 X 的影响传递到那些和 X 直接相关的变量上。这部分是将所有包含 X 的关系进行连接操作(第 13 行), 然后将 X 投影出结果关系(第 14 行)。这样就构造出一个简化的问题(少了一个变量), 且这个问题可以被递归解决。为了得到 X 的可能取值, 算法将具有关系 R (该关系定义了 X 的影响)的简化问题的解进行连接。

如果仅仅想得到一个解, 那么算法将返回连接操作后的一个元素, 而不是返回 $R \bowtie S$ 。不论算法返回了哪个元素, 这个元素都被确保是解的一部分。如果算法中 R 的任何值都不包含元组, 那么说明无解。

VE 算法的有效性取决于变量选择的顺序。中间结构(在变量的中间关系上体现)不取决于关系的实际内容, 但是取决于整个网络的图形结构。算法的有效性可以通过考虑图形化结构来决定。通常, VE 算法在约束网络稀疏的情况下是有效的。对于一个特定的变量顺序, 返回最大关系下的变量数可以称之为这个变量顺序下图形的树宽(treewidth)。一个

```

1: procedure  $VE\_CSP(V, C)$ 
2:   Inputs
3:      $V$ : 变量集合
4:      $C$ :  $V$  上的约束集合
5:   Output
6:     包含所有一致的变量赋值的关系
7:   if  $V$  仅包含一个元素 then
8:     返回  $C$  中所有关系的连接
9:   else
10:    select 变量  $X$ , 并删除
11:     $V' := V \setminus \{X\}$ 
12:     $C_x := \{T \in C: T \text{ 涉及 } X\}$ 
13:    let  $R$  是  $C_x$  中所有约束的连接
14:    let  $R'$  是  $R$  投影到非  $X$  变量的结果
15:     $S := VE\_CSP(V', (C \setminus C_x) \cup \{R'\})$ 
16:    return  $R \bowtie S$ 

```

图 4-5 CSP 问题的变量消除法

图形的树宽则是任意顺序下最小树宽。VE 的时间复杂度在树宽上是指数的，在变量数上是线性的。本章 4.4 节中的搜索方法在变量数上就是指数的。

选择一个消除顺序从而产生最小树宽是一个 NP 难问题。尽管如此，还是存在一些好的启发式条件，最常见的两个是：

- **最小因式(min-factor)**：在每个阶段，选择能产生最小关系的变量。
- **最小缺损(minimum deficiency)或最小填充(minimum fill)**：在每个阶段，选择能使剩余的约束网络添加最小数量弧的变量。和 X 在一个关系中，且不和其他变量在一个关系中的变量的数称为变量 X 的缺失。这可以理解为如下含义：对于能产生大关系的变量，只要确保不使网络更复杂，那么是可以删除它的。

对比最小因式条件，最小缺损常常能被经验性地找到，从而给出一个更小的树宽，但是更难计算。

VE 方法也可以和弧一致方法组合使用，每当 VE 删除一个变量时，弧一致方法都可以进一步简化网络。这种方式可以产生较小的中间表格。

4.8 局部搜索

之前各节研究的算法都是系统地搜索了整个空间。如果空间是有限的，那么算法或者找到一个解，或者告知无解。但是一些搜索空间对于系统搜索来说过于庞大，甚至有可能是一个无限的空间。而在合理的时间范围内，系统搜索不能遍历整个空间并给出一个有意义的结果。接下来本节阐述的方法将试图在这些非常庞大的空间内运行。这些方法不是系统地搜索整个搜索空间，它们被设计成在一个平均范围内快速地寻找解。这类方法不能肯定保证解被找到，或者肯定存在一个解，甚至不能证明问题无解。对于那些已经知道有解或者可能有解的应用往往选择这类方法。

局部搜索方法开始时先完整地赋值，即对每一个变量赋一个的值，进而试图迭代地提升这套赋值质量，提升的手段包括阶梯性提升、随机性选取或者重新选择一套赋值。当前提出了多种局部搜索策略。面对运筹学和人工智能中的不同问题，这些方法何时生效成为学者们研究的焦点。

针对 CSP 问题图 4-6 给出了通用的局部搜索算法。 A 代表对每个变量赋一个值。每次循环的开始为每个变量赋随机值。这种赋值第一次执行叫做**随机初始化(random initialization)**。外部循环的每次迭代称为一次**尝试(try)**。实施一个新尝试的通用方法就是执行一次**随机重启(random restart)**。

一种代替随机初始化的方法是使用一个构造启发式条件来猜测一个解，然后迭代地改进。

在赋值空间内 while 循环里执行的是一个**局部搜索(local search)**，或者说是一次**游走**

```

1: procedure Local-Search( $V, dom, C$ )
2:   Inputs
3:      $V$ : 变量集合
4:      $dom$ : 函数，函数  $dom(X)$  为变量  $X$  的域
5:      $C$ :  $V$  上的约束集合
6:   Output
7:     满足约束的完整赋值
8:   Local
9:      $A[V]$  是被  $V$  索引的一个变量值数组
10:  repeat
11:    for each 变量  $X$  do
12:       $A[X] \leftarrow dom(X)$  中的一个随机值
13:    while 不满足终止条件且  $A$  不是一个满意的赋值 do
14:      选择一个变量  $Y$  并且选择一个值  $V, V \in dom(Y)$ 
15:      设置  $A[Y] \leftarrow V$ 
16:      if  $A$  是一个满意的赋值 then
17:        return  $A$ 
18:  until 终止
  
```

图 4-6 为 CSP 问题寻找一个解的局部搜索

(walk)。对当前赋值 S ，考虑其邻居(neighbor)，并从中选择一个邻居作为下一次的当前赋值。在图 4-6 中，一个全赋值的邻居是那些与全赋值的某一单个变量值不同的赋值。针对邻居集合，不同的选择会产生不同的搜索算法。

在赋值上的局部搜索一直执行直到找到满意的赋值为止，或者是执行到终止条件被满足。终止条件决定了何时停止当前的局部搜索并且执行一个随机重启，即用新的赋值来重新启动。对于终止条件，最简单的设定就是规定其执行确定次数的循环。

该算法不能保证一定会停止。如果无解，那么算法将永远执行下去。算法也很可能在搜索空间的边缘被困住。一个算法每当有解的时候总能找到一个答案，那么这个算法是完备的(complete)。因此上述算法是不完备的。

131

该算法的一个实例是随机采样(random sampling)。随机采样算法将 while 内的终止条件置为真，因此内部 while 循环将不会被执行。随机采样将在外部循环不断地选取随机赋值直到寻找到一个满足约束的赋值，否则算法不会停止。随机采样的方法从意义上来讲是完备的，只要给定充足的时间，如果有解存在时能确保找到一个解，但是在时间上却没有上界。算法效率低。算法的效率仅取决于域大小和解个数这两个元素的乘积。

算法的另一个实例是随机游走(random walk)。在这个算法中，只有找到一个满意的赋值(即，while 内终止条件总为假且没有随机重启)时，才会跳出 while 循环。在 while 循环内，算法随机地选择一个变量和一个值。从意义上来讲，随机游走和随机采样是相同的，都是完备的。算法在每步采用的时间都少于为所有变量重新采样，但是却比随机采样消耗更多步，主要取决于解的分布。当变量域的大小不同时，需要应用随机游走算法的各种变体。随机游走算法可以随机选择一个变量然后随机选择一个值，也可以成对地随机选择变量和值。当有一个较大的域时，更可能选择算法的后一种变体。

4.8.1 迭代最佳改进

在迭代最佳改进(iterative best improvement)方法中，当前被选择节点的邻居节点是能够优化评估函数(evaluation function)的节点。在贪婪下降(greedy descent)法中，被选择的邻居节点是能最小化评估函数的节点。当目标最大化时，这也被称为爬山(hill climbing)算法或贪婪上升(greedy ascent)法。通常我们只考虑最小化，如果想最大化一个量，可以选择最小化该量的“否”。

迭代最佳改进需要一个评估每个整体赋值的方法。对于 CSP 问题，一个常见的评估函数就是最小化在全赋值下违反的约束的数量。一个被违反的约束可以称为一个冲突(conflict)。当评估函数是冲突数时，解则为使得评估为零的一个全赋值。有时也通过一些约束的权重来重定义评估函数。

局部最优(local optimum)是一个赋值，在该赋值下的邻居节点不能够提升评估函数。在贪婪下降法中，局部最优就是局部最小；在贪婪上升法中，局部最优就是局部最大。

【例 4-25】在例 4-8 配送时序安排问题中。假设梯度下降初始于这样的赋值： $A=2$ ， $B=2$ ， $C=3$ ， $D=2$ ， $E=1$ 。因为它不满足 $A \neq B$ ， $B \neq D$ 和 $C < D$ ，所以对这个赋值产生一个为 3 的评估。用最小评估得到了它的邻居节点 $B=4$ ，此时因为只有 $C < D$ 没有被满足，因此产生了为 1 的评估。此时已经是局部最小。接下来一个随机游走将 D 变为 4，这时产生了为 2 的评估。将 A 变为 4，产生一个为 2 的评估，将 B 变为 2，产生了为零的评估。这样就找到了一个解。

132

下面给出通过游走得到的赋值轨迹：

A	B	C	D	E	评估
2	2	3	2	1	3
2	4	3	2	1	1
2	4	3	4	1	2
4	4	3	4	1	2
4	2	3	4	1	0

当多个赋值可以得到相同的评估时，不同的初始化或者不同的选择将给出不同的结果。◀

如果变量有小的有限域，当选择邻居时，局部搜索可以考虑变量其他所有的值。如果变量的域很大，考虑其余所有的值则代价会很高。一个替换的策略是只考虑一个变量的一部分的变量值，通常是接近的值。有时为了选择一个替换值，会使用一些颇具匠心的方法。

典型的局部搜索方法是考虑当前赋值的最相邻的赋值，即使这个赋值与当前赋值是相等的或者是更差的。通常快速地做出选择比花大量时间做出最佳选择的效果更好。对邻居的选择有多种变化方式：

- 1) 将变量和值一起选择。在任意变量的所有不同赋值中，选择一个能使评估函数最小化的赋值。如果能使得评估函数最小化的赋值多余一个，那么在它们中随机选择一个。
- 2) 先选择一个变量，然后选择它的值。对于选择变量来说，有很多种可能性：
 - 掌握每个变量涉及了多少个不被满足的约束，选择一个涉及了最多不满足的约束的变量。
 - 随机选择一个涉及了任意不被满足的约束的变量。
 - 随机选择一个变量。

一旦变量选定后，算法就可以选择变量值了，这个变量值既可以是产生最佳评估值的值，也可以是随机选取的一个值。

- 3) 随机选取变量或者变量值，如果能提升评估，那么就接受这个改变。

这三种选择中，成对地优化变量-变量值比其他两种有更好的效果，但是需要花费更多的计算。尽管一些理论性结果存在，但是在实际中为了更好地运行，选择哪种方式则是经验问题。

4.8.2 随机算法

迭代最佳改进对当前赋值随机地选择了最好邻居集中的一个邻居，也可以用以下两种随机方式来跳出局部最小值：

- 随机重启，所有变量的值被随机选择。这使得搜索从搜索空间内的一个完全不同的部分再次开始。
- 随机游走，采用一些随机移动，其中插入一些优化移动。在贪婪下降中，该操作允许上升的移动，这样可以使随机游走跳出局部最小值。

随机游走是一个局部随机移动，而随机重启是一个全局随机移动。当问题涉及很多变量时，随机重启消耗代价大。

利用随机移动的一种混合贪婪下降法是一类算法的实例，这类算法称为**随机局部搜索** (stochastic local search)。

但因搜索空间常常有成千上万的维度，且每一维都有一个离散值的集合，所以很难将算法执行的搜索空间可视化。一些直觉的结果可以从低维问题中收集到。考虑图 4-7 中的

两个一维搜索空间，在图中想要得到最小值。假设可以通过对当前位置进行向左或向右的一个小的移动来获得邻居位置。为了在搜索空间(a)内寻找全局最小的值，我们希望利用随机重启的贪婪下降法来快速获得这个最优值。一旦随机选择找到了最深的谷中的一点，那么贪婪下降将会快速趋向于全局最小。因为许多随机移动都被要求跳出当前的一个局部极值，所以随机游走法在搜索空间(a)中就不会有很好的效果。但是，随机重启在搜索空间(b)中就会被快速地困在锯齿的峰值上，且不能得到好的结果。但是，贪婪下降和随机游走的组合就可以跳出这些局部极值，而且使用一次移动或者两次移动可能就足够跳出这个局部极值。因此，随机游走在搜索空间(b)中就有很好的效果。

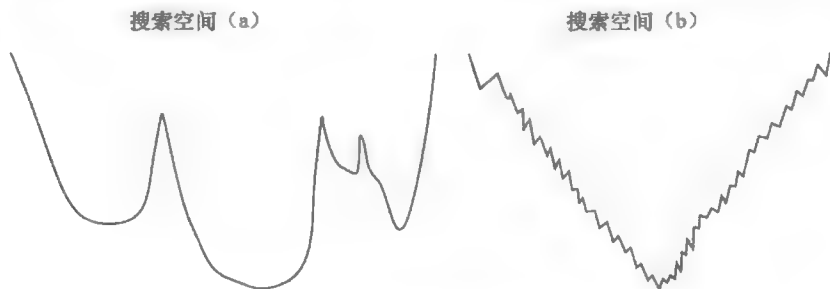


图 4-7 两个寻找最小值的搜索空间

以上呈现的局部搜索是没有存储的，在执行的过程中不记录搜索行为。用存储来提升局部搜索的一种简单方法就是禁忌表(tabu list)，这个禁忌表中存有最近被更改过的变量-变量值。禁忌表的思想是当选择一个新的赋值时，不得选择禁忌表中的变量-变量值。该方法阻止了在一些赋值中不断循环。禁忌表的大小一般是一个小的固定值，而禁忌表的大小也是可以优化的参数。当禁忌表大小为 1 时，等价于不允许同样的赋值被立即重新访问。

各种算法的不同之处在于它们采用了多少工作量来确保最佳改进。举一个极端的情况，算法在选择新赋值时可以确保给出的是相对于所有邻居的最佳改进。另一个极端的情况是，算法随机地选取一个新的赋值，并且拒绝使情况更糟的赋值。本节的剩余部分将给出一些典型算法，各种算法的不同之处在于它们付出了多少计算工作量来确保完成最佳改进。哪个方法能取得最好的效果通常是一个经验问题。

1. 常用的改进手段

第一种方法总是成对地选择变量-变量值来完成最佳改进。最朴素的办法就是线性地扫描每个变量和它的每个变量值，然后对比当前的所有变量的赋值决定哪个赋值能使得不被满足的约束数量最少，进而选择一个能产生最佳改进的变量-变量值对，即使这个改进是有负面影响的。当变量数为 n ，平均域大小为 d ，一个赋值的邻居数为 r 时，这一步的复杂度是 $O(ndr)$ 。

一个更精细的替换策略是设置一个变量-变量值的优先队列。对于任意的变量 X ，值 v 在 X 的域内，并且在当前赋值下变量 X 没有被赋值为 v ，那么 $\langle X, v \rangle$ 对就将被填入优先队列。 $\langle X, v \rangle$ 的权重为当前完整赋值的评估减去将 X 替换为 v 的完整赋值的评估。也就是说，它包含了每个替换值的评估变化。在每个阶段，算法选择一个最小权重值，这个值则是邻居所能提供的最佳改进。

一旦变量 X 被赋予一个新值，那么所有变量中有多个权重将被改变，必须重新计算后插入优先队列中。当 X 被赋予 v 值时，被更改权重的那些变量是由于这个新值而出现在一个被满足或不被满足约束中的变量。

当变量数为 n ，平均域大小为 d ，一个赋值的邻居数为 r 时，算法这一步的复杂度是 $O(\log nd + rd \log nd)$ 。

由于算法在每次都要确保最大的改进，因此在掌握数据的结构方面花费了太多时间。

2. 两阶段选择

一个替换策略是将成对选择变量-变量值策略进行分割，首先选择一个变量进行更改，然后再选择一个值。

该算法也有一个变量优先队列，队列中变量的权重为变量所参与的冲突约束数。每次算法都选择一个具有最大权重的变量。一旦变量被选中，那么这个变量将被赋予一个能最小化约束冲突数的值。作为最新赋值的一个结果，每个变量的约束冲突的值也会发生改变，其余参与到这个冲突的变量的权重也需要被更改。

算法这一步的复杂度是 $O(\log n + r \log n)$ 。对比成对地选择最佳变量-变量值方法，该算法在每次移动中做的工作量较小，但在固定时间间隔内需要完成更多的移动。尽管这些移动倾向于完成较小的改进，但是在移动数和每次移动的复杂度两者间的权衡需要经验性的评估。

3. 任意冲突

相对于选择最佳的移动，一个更为简单的替换策略是选择参与约束冲突的任意变量，并改变它的值。在每一次移动随机任意选择一个参与到不满足约束的变量。该算法将赋予这个变量一个能最小化约束冲突数的值。

为了完成这个替换策略，我们需要一个数据结构，用这个结构来表示涉及一个约束冲突的变量集合 C 。该数据结构应被设计为可以快速选择 C 中的一个随机成员。当 X 的值发生改变时，每个包含 X 的约束都被检查。对于每个变为不被满足的约束，所有包含在该约束中的变量都被添加到 C 中。对于每个变为满足的约束，存在包含在该约束内且不包含在另一个不被满足的约束中的变量，将这些变量从 C 中删除。检查一个变量是否包含在另一个冲突约束中的方法是掌握每个变量的一组冲突数，这个数在一个约束变为不被满足时将增加，在约束变为被满足时将减少。

可以将随机移动、随机重启和禁忌表机制与上述算法组合使用。

4. 模拟退火

最后一种方法将不再获取约束冲突数据的结构，取而代之的是随机地选择一个邻居，然后对于新赋值选择接受还是拒绝。

退火是冶金术中的一个过程，在金属慢慢冷却的过程中，使它们达到一个低能量的状态。模拟退火是一种类似最优化的方法。它用热力学的术语来描述。随机运动对应了高温；在低温条件下则有很小的随机性。模拟退火的执行过程是从在高温下进行随机搜索开始，然后温度慢慢降低，最后到达零度时变成纯贪婪下降。随机性应该倾向于跳出局部极值，并且找到一个低启发值作为邻域；贪婪下降的过程中将导致局部最小。高温条件比低温条件更有可能导致逐步恶化。

模拟退火掌握了当前变量的一个赋值情况。在每一移动中，随机选取一个变量，并且随机选取一个值。如果对该变量的赋值是一个改进或者它没有增加约束冲突数，那么算法接受这个赋值，并依此作为当前的新赋值。另外，算法概率性的接受赋值，这取决于当前的温度和新赋值比当前赋值差了多少。如果当前的改变不被接受，那么当前赋值也不会被改变。

为了控制变化到什么程度是可以被接受的，设定一个正实值的温度 T 。假设 A 是当前的一套赋值。假设 $h(A)$ 是 A 被最小化的评估值。为了解决约束问题， h 通常为约束冲突

134

}

135

136

数。模拟退火随机地选择一个邻居，给出新的赋值 A' 。如果 $h(A') \leq h(A)$ ，算法接受赋值 A' ，并把其作为新的赋值。否则，这个赋值仅以下述这个概率被随机接受：

$$e^{(h(A) - h(A'))/T}$$

这样，如果 $h(A')$ 与 $h(A)$ 很接近，那么被接受的可能性会更大。如果温度很高，那么指数趋近于 0，接受概率则接近 1。当温度接近 0 时，指数接近 $-\infty$ ，且接受概率接近 0。

137

图 4-8 中给出了不同温度下逐步变差的接受概率。图中， k 变差表示 $h(A') - h(A) = k$ 。例如，如果温度为 10 ($T=10$)，改变为 1-worse ($h(a) - h(a') = -1$) 的接受概率是 $e^{-0.1} \approx 0.9$ 。改变为 2-worse 的接受概率为 $e^{-0.2} \approx 0.82$ 。如果温度是 1，改变为 1-worse 的接受概率是 $e^{-1} \approx 0.37$ 。如果温度是 0.1，改变为 1-worse 的接受概率是 $e^{-10} \approx 0.000\ 05$ 。这个温度下，它本质上仅是一个提升赋值或维持不变的操作。

如果像 $T=10$ 这种情况，温度很高时，算法倾向于接受变差程度很小的移动；不倾向于接受变差程度很大的移动。算法对于改进的移动有轻微的偏好。当温度减少时（例如，当 $T=1$ 时），尽管变化的移动是有可能的，但是可能性较小。当温度非常小时（例如，当 $T=0.1$ 时），选择一个变差的移动将非常罕见。

模拟退火需要一个退火计划 (annealing schedule)，这个计划说明在搜索的过程中，温度是怎样下降的。几何冷却法是一个被广泛应用的计划。例如，几何冷却计划开始于温度 10，且每次移动乘以 0.97，在 100 次移动后温度将变为 0.48。找到一个好的退火计划是一种艺术。

温度	接受概率		
	1-worse	2-worse	3-worse
10	0.9	0.82	0.74
1	0.37	0.14	0.05
0.25	0.018	0.000 3	0.000 006
0.1	0.000 05	2×10^{-9}	9×10^{-14}

图 4-8 模拟退火接受更差移动的概率

4.8.3 评估随机算法

当随机算法给出一个不同的结果和一个不同的运行时间时，即使是对同样的问题，这些算法也很难相互比较。当有时算法不能找到一个解时，比较起来就显得更为困难，这时算法要么永远运行下去，要么终止于一个任意位置。

不幸的是，像运行时间的平均值或中位值，这类汇总统计在比较算法上并不是很有效。例如，如果想用运行时间的平均值来比较算法，那么就必须考虑算法运行失败的时候（没有找到解）怎样计算平均值。如果在计算平均值上忽略这些运行失败的情况，那么随机地选择一个赋值然后就停止，这类算法将成为最好的算法，因为这类方法几乎都失败了，但是一旦成功了，速度是最快的。如果将无法停止的算法算成是无限时间，那么所有算法都存在寻找不到解的情况，则变成算法平均运行时间都是无限的。如果把规定停止时间作为无停止运行的时间，尽管允许在更快地找到一些解和找到更多的解之间寻求一种基本的权衡，但以此方式使用均值的评价更多是针对停止时间的函数，而不是算法本身。

如果用运行时间的中位数来比较算法，那么一个算法以非常慢的速度完成问题的 51% 会比另一个以非常快的速度完成问题的 49% 得到更好的评价，然而事实上后一个算法更有

作用。这个问题在于中位数(50%)是一个随机的值。当然也可以考虑 47% 或者 87% 的值。

对于一个实际问题,可视化算法运行时间的一个方法是使用运行时间分布图(run-time distribution)。该分布在单个问题上能够体现随机算法在运行时间方面的变化性。在 x 轴上既可以标注为移动的次数,也可以标注为运行的时间。对于 x 轴上的每个值, y 轴表示在运行时间内或移动次数内,算法解决问题的次数或解决的比例。因此,它提供了一个累积的分布,这个分布展示了在某些移动数上或者某些运行时间上问题多长时间被解决一次。例如,可以通过 y 轴刻度的 30% 映射到 x 轴上的值,从而找到运行了 30% 的运行时间。运行时间分布图可以通过多次运行算法绘制(或近似)出来(可以说运行 100 次是粗糙近似,1000 次是合理准确地绘制),然后用运行时间进行排序。

138

【例 4-26】 图 4-9 给出了对于单个问题 4 种经验性的随机算法产生的运行时间分布图。 x 轴以对数指标表示了算法移动次数。 y 轴表示在 1000 次中当前问题被成功解决的实例数。图展现了在同一问题上的 4 个运行分布图。算法 1 和 2 在第 10 步或者更少移动的情况下完成了问题的 40%。算法 3 在执行到 10 或者更少移动的情况下完成了问题的 50%。算法 4 在执行到 10 或者更少移动的情况下完成了问题的 12%。算法 1 和 2 在 58% 的时间内能找到一个解,算法 3 在 80% 的时间内能找到一个解,而算法 4 总能找到一个解。这仅仅比较了移动的次数。所耗时间会是更好的度量方式,但是对于小问题来说,度量起来会比较困难,并且还取决于一些具体的手段。

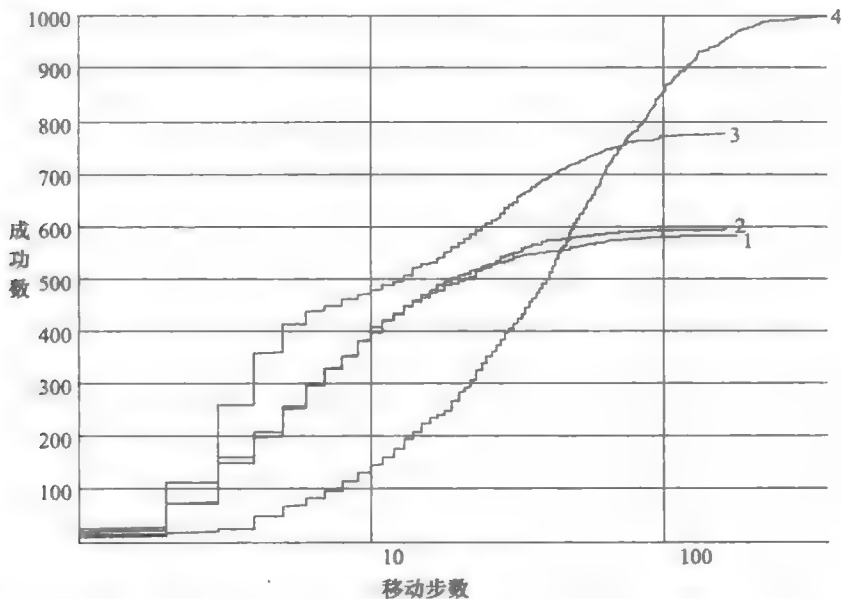


图 4-9 运行时间分布图。这是运行 1000 次的运行时间分布图,每次运行限制在 1000 次移动以内。 x 轴以对数指标表示了算法移动次数, y 轴表示 1000 次中成功解决数。这个是 CSP 样例中 Alsplace.org 的“调度问题 1”。分布图 1 和 2 是两阶段贪婪下降法的两次独立的运行。算法 3 是一阶段贪婪下降法。算法 4 是使用随机行走的贪婪下降法,选择的开始节点是参与到约束冲突的节点(在 Alsplace.org 内的红色节点),变量的最佳值以 50% 的概率被选择,否则选择一个随机值

如果一个算法的运行时间分布图始终处于另一算法分布图的左上方,那么就说第一个算法严格支配第二个算法。这种度量方式下,往往两个算法是不可比的。哪个算法更好取决于所能接受的运行时间和找到一个解的重要性。

在算法执行了一定量的移动后,我们则可以用运行时间分布图去预测若使用随机重启

后该算法的效果。直观地说,在重启发生时,随机重启将重复并适当缩减分布图的左下角。如果运行时间足够,在执行了一定量贪婪下降的移动后,使用随机重启的策略可以将那行有时能找到解的算法变成总是能找到解的算法。

【例 4-27】图 4-9 中,算法 3 支配算法 1 和 2。算法 1 和 2 实际上是相同方法在相同设置下对不同集合的执行。这也是相同的随机算法在相同实例上多重运行的典型错误。算法 3 在移动为 60 之前都优于算法 4,但之后算法 4 较好。

观察图片,算法 3 常常在它最先的 4 或 5 次移动时等到解,之后就没那么有效。这使我们想到,可在算法 3 的每 5 次移动后设置随机重启(每次重启算一次移动)。这么做也确实使得算法 3 对于该问题实例在一定移动数下支配其他所有算法。尽管如此,因为随机重启是代价较高的操作,所以该算法未必是最有效的,也没有必要去预测该算法在其他实例问题上的效果。

4.8.4 局部搜索中利用命题结构

如果 CSP 问题是命题可满足问题,这些问题由二元变量和约束子句构成,那么随机局部搜索处理起来会更加简单。因为以下三个原因,局部搜索的每一步都会变得更加有效:

- 因为对于每套赋值的每个变量,仅有一个替换值存在,所以算法不必去搜索替换值。
- 在不被满足的子句中改变任何一个值都使得子句变为被满足的。这就使得子句很容易被满足,但是这可能导致其他子句变为不被满足的。
- 如果一个变量被更改为 *true*,那么只要包含它的否的那些子句都不被满足。类似推断变量值变为 *false*。这使子句的索引可以非常快速。

任意有限的 CSP 问题都有可能被转化为一个命题可满足问题。具有域 $\{v_1, \dots, v_k\}$ 的变量 Y 可以被转化为 k 个二元变量 $\{Y_1, \dots, Y_k\}$, 当 Y_i 为真时表示 Y 取值 v_i , 取其他值时为假。每个 Y_i 可以被称为一个指示变量(indicator variable)。在每个约束中针对每个为假的元组都有一个子句,该子句指出 Y_i 的哪个赋值是不满足约束的。有的约束也指出在 $i \neq j$ 时, Y_i 和 Y_j 不能同时为真。也有约束可能要求变量 Y_i 必须为 *true*。将 CSP 问题转化为命题可满足问题有三点潜在优势:

- 每个局部移动可以更简单,因此执行起来更为有效。
- 搜索空间被扩展。实际上在找到一个解之前,可以不止一个 Y_i 为 *true* (对应了变量 Y 有多个值)或者可以所有 Y_i 都为 *false* (对应了变量 Y 为空)。这就是说,在原问题中的局部最小的赋值,在新问题中就可能不再是局部最小。
- 相比其他类型的 CSP 问题,命题可满足性被更广泛地研究。由于研究者针对命题可满足做了更多的算法研究,所以当前有更多有效的解决办法。

在一个实际问题中,这种转换是否能令搜索有更好的效果同样也是一个经验问题。

4.9 基于种群的方法

之前的局部搜索算法都是维护当前一个单独的赋值方案。本节研究的算法将维护多个赋值方案。第一种方法是集束搜索,维护了最佳的 k 个赋值方案。接下来的随机集束搜索将概率性地随机选择多个赋值方案。在受生物进化启发的遗传算法中, k 个赋值方案组成

了一个相互作用的种群，使用各种方法将该种群进化到一个新的种群。在这些算法中，对每个变量都赋一个值的全赋值被称为个体(individual)，且当前个体的集合称为种群(population)。

集束搜索(beam search)是一个类似于迭代最佳改进的方法，但是该方法不是维护一个赋值，而是同时维护 k 个赋值。当算法找到一个满足约束的赋值时，则会报告成功。算法的每个阶段都会选择当前个体的 k 个最佳邻居(当少于 k 个时则将它们全部选中)且同时随机选择。算法使用新的 k 个赋值方案的集合重复执行。

集束搜索同时考虑了多个赋值方案。该算法对存储有界的案例比较有用， k 的选择取决于可用的存储空间。

随机集束搜索(stochastic beam search)是集束搜索的一个替换策略，不适用选择最佳 k 个个体的策略，而是随机地选择 k 个个体，更倾向于选择能得到更好评估的个体。这样的选择需要两个函数，一个是概率选择函数，另一个是评估函数。标准的方法是使用吉布斯分布(Gibbs distribution)或者波尔兹曼分布(Boltzmann distribution)，用如下的概率比例去选择一个赋值方案 A ：

$$e^{-h(A)/T}$$

这里 $h(A)$ 是评估函数， T 是温度。

随机集束搜索倾向于允许在 k 个个体中具有更多差异性，而不是普通的集束搜索。用生物进化学的术语，评估函数反映了个体的适应度；越适应的个体就越有可能被通过，这意味着这个赋值方案的一部分在下一种群中是好的。随机集束搜索就像一个无性繁殖的过程，每个个体给出了具有少量变异的孩子，然后随机集束搜索处理这些具有最佳适应度的存活个体。需要注意的是，在随机集束搜索中，一个个体可能被随机地多次选择。

遗传算法(genetic algorithm)进一步完善了类演化的过程。它和随机集束搜索很像，但是种群中的每个元素是有区别的。在遗传算法中，种群内的每个新元素变为个体-个体父代的成对组合。尤其是遗传算法选择成对的个体，针对子代变量的值，某些值从其中一个父代选取，剩下的值从另一个父代选取，进而创作一个新的个体。这一过程有点类似于有性繁殖的 DNA 的拼接过程。

遗传算法中，这一新的操作被称为**交叉**(crossover)。均匀交叉是选择两个个体(双亲)，创造两个新个体称为子代(children)。在一个子代中，每个变量的值都来自双亲中的一个。一个普通的方法是**单点交叉**(one-point crossover)，该方法假设变量有一个总体的排序。一个索引 i 被随机地选择，然后在 i 之前的变量的值从一个父代中选取， i 之后的变量的值从另一个父代选取，用这样的方式来构建子代中的一个孩子。该算法交叉操作的有效性取决于变量的整体排序。变量的排序也是设计遗传算法的一部分工作。

假设有一个 k 个(k 是偶数)个体的种群。一个基本的遗传算法需要维护 k 个个体(k 个个体共同作为一代)，并且使用这 k 个个体，通过如下步骤产生新一代：

- 随机选择个体对，更能适应的个体被选择的可能性更大。一个更适应的个体比另一个个体多出多少被选择的可能性这主要取决于不同的适应度标准和一个温度参数。
 - 对每一个体对执行一次交叉。
 - 对于一些随机选择的变量，通过选择一些其他的值对其进行随机变异(非常少)。
- 这是一个随机游走的移动操作。

延续这种处理直到创建了 k 个个体，并且对下一代执行该操作，算法在图 4-10 中给出。

【例 4-28】 例 4-8 中，假设我们使用了和例 4-25 一样的评估函数，也就是不被满足的约束数。个体 $A=2, B=2, C=3, D=1, E=1$ 的评估值为 4。形成这个较低的评估（较少不被满足的约束），主要是因为 $E=1$ 。那么保留这个值的子代将比不保留这个值的子代更有可能获得更低的评估值，更有可能存活（评估值越小代表被满足的约束越多）。其他个体获得较低评估值的原因可能就不同，例如，个体 $A=4, B=2, C=3, D=4, E=4$ 的评估值同样为 4，这主要是由于前四个变量的赋值。同样的，子代保留这个性质比不保留更有适应能力，更可能存活。如果这两个个体进行配对，那么子代中的一些个体将会继承两者的劣质属性，并且将死亡。子代中的一些个体也将会偶然地继承两者的优秀属性，将有较高的存活机会。

正如在其他的随机局部搜索算法中一样，设计特征和评估函数是困难的，因此算法不能摆脱局部极值。算法的有效性对描述问题的变量和变量顺序敏感。这一研究工作也是门艺术。在其他的许多启发式算法中，进化算法有很大的自由度，因此针对良好性方面比较难以配置和调整。而且类似自然演化的过程容易造成误导性，因为自然界演化对组合决策和最优化问题给出的并不总是最佳策略。

大量研究人员研究了遗传算法在真实问题上的实用性，并得出了一些有价值的结论。本节所描述的只是遗传算法的一种。

4.10 最优化

通常我们不是仅仅考虑存在或不存在可能世界可以满足约束，而是想得到一个最好的可能世界。在可能世界中经常存在一个偏好 (preference) 关系，我们需要对应偏好得到最好的可能世界。这个偏好经常是最小化某些误差。

给出一个最优化问题 (optimization problem)：

- 一个变量集，每个变量有关联的域；
- 一个目标函数 (objective function)，该目标函数将赋值方案映射到一个实数；

```

1: procedure GeneticAlgorithm( $V, dom, C, S, k$ )
2:   Inputs
3:    $V$ : 变量集合
4:    $dom$ : 函数，函数  $dom(X)$  为变量  $X$  的域
5:    $C$ :  $V$  上的约束集合
6:    $S$ : 针对温度的退火计划
7:    $k$ : 种群大小——一个偶数
8:   Output
9:   满足约束的完整赋值
10:  Local
11:    $Pop$ : 赋值集合
12:    $T$ : real
13:    $Pop \leftarrow k$  个完整的随机赋值
14:    $T$  是根据  $S$  被赋予的值
15:  repeat
16:    if 一些  $A \in Pop$  满足  $C$  中的所有约束 then
17:      return  $A$ 
18:     $Npop \leftarrow \{\}$ 
19:    repeat  $k/2$  times
20:       $A_1 \leftarrow RandomSelection(Pop, T)$ 
21:       $A_2 \leftarrow RandomSelection(Pop, T)$ 
22:       $N_1, N_2 \leftarrow Combine(A_1, A_2)$ 
23:       $Npop \leftarrow Npop \cup \{mutate(N_1), mutate(N_2)\}$ 
24:     $Pop \leftarrow Npop$ 
25:    根据  $S$  更新  $T$ 
26:  until 终止
27: procedure RandomSelection( $Pop, T$ )
28:  从  $Pop$  中以  $e^{-k(A)/T}$  的概率选择  $A$ 
29:  return  $A$ 
30: procedure Combine( $A_1, A_2$ )
31:  随机选择整数  $i, 1 \leq i < |V|$ 
32:  令  $N_1 \leftarrow \{(X_i = v_i) \in A_1, j \leq i\} \cup \{(X_j = v_j) \in A_2, j > i\}$ 
33:  令  $N_2 \leftarrow \{(X_j = v_j) \in A_2, j \leq i\} \cup \{(X_i = v_i) \in A_1, j > i\}$ 
34:  return  $N_1, N_2$ 
  
```

图 4-10 为 CSP 问题寻找一个解的遗传算法

- 一个**最优化标准**(optimality criterion), 用该准则找到一个能最大化或最小化目标函数的赋值方案。

这说明要根据最优化标准来找到一个最佳的赋值方案。例如, 最优化标准就是最小化目标函数。

约束优化问题(constrained optimization problem)是具有强约束的最优化问题, 这些强约束指明了变量的赋值是否可能。也就是说, 需要找到一个最好的赋值, 该赋值满足硬约束。

144

已经有大量的文献对最优化进行了研究。对于实际的有约束最优化问题, 也存在了很多技术。例如, 线性规划是一类有约束最优化技术, 这里变量都是实值的, 目标函数是变量组成的一个线性函数, 并且硬约束是一些线性不等式。这里我们不详细阐述这些具体的技术。尽管这些技术都有实际应用, 但是在所有优化问题的空间内还是适应性有限。我们只说明一些通用技术, 这些技术允许更多的通用目标函数。然而, 要解决的问题恰好是某些具体算法中的一类, 那么用这些具体的技术会比用这里提到的通用技术效果更好。

在**约束优化问题**中, 目标函数被分解为变量子集的函数集, 称为**软约束**(soft constraint)。软约束为每个变量子集的赋值方案分配一个成本。赋值方案的目标函数值是对给定软约束成本的和。典型的最优化标准是将目标函数最小化。

与硬约束一样, 软约束是一个变量集的作用域(scope)。软约束是将作用域内变量的域映射为实数的一个函数, 称之为**评估**(evaluation)。因此, 对于作用域内变量的赋值方案, 该函数返回的是一个实数值。

【例 4-29】 同例 4-8 一样, 假设必须调度一组配送活动。但是不同于硬约束, 本例中在配送次数上有偏好。与配送次数组合关联的成本就是本例中的软约束。这说明在本例中要找到一个最小成本和的配送计划。

假设 A 、 C 、 D 、 E 四个变量的域都是 $\{1, 2\}$, 且变量 B 的域为 $\{1, 2, 3\}$ 。软约束如下:

c_1 :	A	B	成本	c_2 :	B	C	成本	c_3 :	B	D	成本
	1	1	5		1	1	5		1	1	3
	1	2	2		1	2	2		1	2	0
	1	3	2		2	1	0		2	1	2
	2	1	0		2	2	4		2	2	2
	2	2	4		3	1	2		3	1	2
	2	3	3		3	2	0		3	2	4

c_1 的作用域是 $\{A, B\}$, c_2 的作用域是 $\{B, C\}$, c_3 的作用域是 $\{B, D\}$ 。假设 c_4 的作用域是 $\{C, E\}$, c_5 的作用域是 $\{D, E\}$ 。

在一个软约束中对一些变量的赋值是其他变量的一个函数。在上例中, 在 $c_1 (A=1)$ 是 B 的一个函数, 当 $B=2$ 时, 函数评估为 2。

给定一个全赋值, 全赋值的评估值就是应用了该全赋值的软约束的评估值加和。一个形式化的方法是在软约束上定义操作符。软约束可以被逐点添加。两个软约束的叠加是将这两个约束的作用域范围进行并操作, 叠加的后的评估值是当前赋值下两个函数值的和。

145

【例 4-30】 考虑前例中的 c_1 和 c_2 函数。 $c_1 + c_2$ 是范围为 $\{A, B, C\}$ 的函数:

$c_1 + c_2$:	A	B	C	成本
	1	1	1	10
	1	1	2	7
	1	2	1	2

第二个值的计算如下:

$$\begin{aligned}
 &(c_1 + c_2)(A=1, B=1, C=2) \\
 &= c_1(A=1, B=1) + c_2(B=1, C=2) \\
 &= 5 + 2 \\
 &= 7
 \end{aligned}$$

与生成-测试算法一致, 寻找最优赋值的一种方法是计算软约束的加和, 选择一个具有虽小值的赋值方案。稍后将介绍其他更有效的最优化算法。

当一个硬约束不被满足时, 硬约束可以被模型化为产生一个无穷大的成本。只要一个赋值的成本是有限的, 那么该赋值就没有违反硬约束条件。如果硬约束不被满足时不用无穷大的成本, 那么另一个策略是将成本设置为一个很大的数, 数要比软约束的加和还大。那么最优化就可以看成是一个寻找解的过程, 这个解违反了最少的硬约束, 并且有一个最小的成本。

最优化问题要难于约束满足问题。很难知道一个赋值方案是不是最优的。在 CSP 问题中, 算法可以判断一个赋值方案是不是一个满足约束的解。而在最优化问题中只能通过和其他赋值方案比较来判断当前赋值方案是不是最优的。

许多解决硬约束的方法都可以被扩展来解决最优化问题, 接下来的小节会进行叙述。

4.10.1 最优化的系统方法

弧一致方法可以通过修剪劣势赋值方案从而被泛化解决最优化问题。假设 c_1, \dots, c_k 是含变量 X 的软约束。设软约束 $c = c_1 + \dots + c_k$ 。设 Y 是包含在约束 c 中不同于 X 的变量。 X 有一个变量值 v , 如果对于 Y 的所有变量值 y , X 的一些值有 $c(X=v', Y=y) < c(X=v, Y=y)$, 对于变量 X 就说 v 是**严格被支配的**(strictly dominated)。修剪严格被支配的值不会导致最优解被删除。可通过 GAC 算法的重复执行对域进行修剪。

被弱支配(weakly dominated)与严格被支配的定义基本相同, 不同之处在于将“小于”替换为“小于等于”。如果仅需要一个解, 那么被弱支配的值可以被相继删除。被弱支配的值的删除可能会影响到寻找最优解的过程, 但是不会删除最优解。从弧一致的角度来看硬约束, 修剪被支配的值(严格的或弱的)可以大大简化问题, 但是它本身并不能解决问题。

域分割法可以用来构建搜索树。域分割选择一些变量 X , 并且考虑变量 X 的每个值。赋给 X 一个值使得包含 X 的约束被简化, 且其他变量的值可以被修剪。实际上, 修剪被弱支配的值意味着当仅有一个变量时, 变量的最佳值可以被计算出来。重复域分割过程建立一个搜索树, 如图 4-1 所示, 但是叶子是评估值。像 A^* 和分支定界这样的搜索算法通过分配确定的成本就可以找到一个最优值。

域分割可以通过两种技术进行改进。第一种,在变量 X 的分割下,对于另外一个赋值不取决于 X 的变量,那么在有 X 变量值的子树中可以共享另一变量的计算,这个值仅需计算一次,可以缓存起来。第二种,如果对一个变量集的删除使得图形被拆分,那么被拆分的部分在变量赋值时可以单独被解决。

变量消除(Variable elimination)是域分割法的一种动态规划变形。变量在某一时刻被删除,一个变量 X 被删除的过程如下:设 R 是包含变量 X 的约束集合。 T 是一个新约束,该约束的作用域是 R 中约束作用域的并,值是 R 中值的和。设 $V = \text{scope}(T) \setminus \{X\}$ 。对 V 中每个变量的值,选择 X 中的一个值来最小化 T ,产生一个新的约束 N , N 的作用域为 V 。约束 N 代替了 R 中的约束。这就产生了一个具有更少变量和新约束集的新问题,该问题可以被递归地解决。约简问题的一个解 S 是 V 中变量的一个赋值方案。因此, $T(S)$ 是 X 的一个函数(赋值 S 下的约束 T)。 X 的最优值则是使 $T(S)$ 为最小值的值。

图 4-11 给出 VE 算法的伪代码。消除顺序可以被先验地或者即时地计算,例如,使用在 CSP VE 中的启发式消除顺序。这个实施过程可以不用存储 T ,仅通过对 N 构建一种扩展的表述。

147

```

1: procedure VE_SC( $V_s, F_s$ )
2:   Inputs
3:      $V_s$ : 变量集合
4:      $F_s$ : 约束集合
5:   Output
6:     对于  $V_s$  的最佳赋值
7:   if  $V_s$  包含一个单独元素或者  $F_s$  包含一个单独的约束 then
8:     令  $F$  是  $F_s$  中约束的加和
9:     return  $F$  中最小值的赋值方案
10:  else
11:    根据一些消除顺序选择  $X, X \in V_s$ 
12:     $R = \{F \in F_s: \text{涉及 } X \text{ 的 } F\}$ 
13:    令  $T$  为  $R$  中约束的加和
14:     $N \leftarrow \min_X T$ 
15:     $S \leftarrow \text{VE\_SC}(V_s \setminus \{X\}, F_s \setminus R \cup \{N\})$ 
16:     $X_{opt} \leftarrow \arg \min_X T(S)$ 
17:    return  $S \cup \{X = X_{opt}\}$ 

```

图 4-11 具有软约束的最优化变量消除

【例 4-31】 考虑例 4-29。首先考虑消除变量 A 。 A 仅出现在约束 $c_1(A, B)$ 。

消除 A 给出 $c_5(B) = \arg \min_A c_1(A, B)$:

B	成本
1	0
2	2
3	2

约束 $c_1(A, B)$ 被 $c_5(B)$ 替代。

假设 B 接下来将被消除。 B 出现在三个约束中 $c_2(B, C)$ 、 $c_3(B, D)$ 和 $c_5(B)$ 。这三个约束相加 $c_2(B, C) + c_3(B, D) + c_5(B)$:

B	C	D	成本
1	1	1	8
1	1	2	5
...			
2	1	1	4
2	1	2	4
...			
3	1	1	6
3	1	2	8

c_2 、 c_3 和 c_6 这三个约束被替换为 $c_7(C, D) = \min_B (c_2(B, C) + c_3(B, D) + c_6(B))$;

C	D	成本
1	1	4
1	2	4
...		

148

现在仍然有三个约束： $c_1(C, E)$ 、 $c_5(D, E)$ 、 $c_7(C, D)$ ，这些被递归优化。假设返回一个解 $C=1$ 、 $D=2$ 、 $E=2$ 。那么使得 $c_2(B, C=1) + c_3(B, D=2) + c_6(B)$ 最小的值为 $B=2$ 。

从 $c_1(A, B)$ 中得到使得 $c_1(A, B=2)$ 最小的值为 $A=1$ 。因此，最佳的解是： $A=1$ ， $B=2$ ， $C=1$ ， $D=2$ ， $E=2$ 。

算法 VE 因为是用硬约束执行的，所以复杂度取决于约束图的结构。在算法 VE 中，也包括算法 VE_SC，稀疏图可能会产生一个很小的中间约束。连接紧密的图会产生繁多的中间约束。

4. 10. 2 局部搜索最优化

局部搜索方法可以直接应用到最优化问题中，用最优化问题中的目标函数作为局部搜索的评估函数。算法在确定的时间内执行(为利用搜索空间的其余部分，也可能使用随机重启的方法)，在这个时间内算法一直寻找最佳赋值，并且返回这个赋值作为答案。

最优化的局部搜索有一个额外的复杂情况，它的结果不能仅仅来源于硬约束，因为很难判断一个全赋值是不是最佳可能解。根据最优化标准，一个**局部最小值**(local minimum)至少和它的邻居一样小。一个**全局最小值**(global minimum)是至少和所有其他全赋值一样小。如果对其他的赋值没有系统的搜索，算法可能不知道目前找到的最佳赋值是不是全局最优，也不知道在搜索空间的另一部分是不是存在一个更好的解。

当同时具有软、硬约束时，解决约束优化问题将成为在违反硬约束和使评估函数更差之间的一个权衡。当违反硬约束时，不会把成本看作是无限制的，因为那样的话算法将区分不出是违反了一个硬约束还是违反了多个硬约束。在搜索的过程中，有时允许临时违反硬约束也不失为一种好的办法。

连续域

对于域为连续值的最优化问题，局部搜索将变得更为复杂。这主要是因为一个赋值方案的邻居节点会很不清晰。结果同样不能仅仅依赖于硬约束，因为这些硬约束往往将空间离散化。

对于最优化问题，**梯度下降**(gradient descent)法被用来寻找最小值，**梯度上升**(gradient ascent)法被用来寻找最大值。梯度下降法像一个游走在下坡的过程，并且总是朝着尽量向下的方向移动。寻找一个赋值方案的邻居的通用方法是按照评估函数 h 的斜率成比例地向下坡移动。因此，梯度下降法在每个方向上成比例地移动，这个比例是该方向偏导数的负数。

149

一个方向上，如果 X 是一个实值变量，当前值为 v ，那么下一次的值应该是：

$$v - \eta \times \left(\frac{dh}{dX} \right)(v)$$

- η 是**步长**(step size)，是一个比例常数，这个步长决定了梯度下降法以多快的速度达到最小值。如果 η 太大，则容易迈过最小值；如果 η 太小，则程序将非常耗时。
- $\frac{dh}{dX}$ 是 h 关于 X 的导数，是关于 X 的一个函数，被 $X=v$ 度量。

【例 4-32】 图 4-12 给出了用单维函数找到局部最小值的一维实例。开始于位置 1。导数是一个较大的正值，所以移动到位置 2。这时导数变为近似为零的负值，选择较小的移动到位置 3。到位置 3 导数为接近零的负值，所以向右更小的移动。也就是说，当寻找到局部最小值时，斜率接近零，采用更小的移动。

对于多维的最优化问题，当有多个变量时，梯度下降法将为在每一维偏导数的基础上做该维的成比例运动。如果 $\langle X_1, \dots, X_n \rangle$ 是必须被赋值的变量，一个赋值方案对应了一个值为 $\langle v_1, \dots, v_n \rangle$ 的元组。假设评估函数 h 是可微的，赋值 $\langle v_1, \dots, v_n \rangle$ 的下一个邻居需要对每一方向都成比例地移动来获取，这个比例是该方向上 h 的斜率。新的 X_i 的值是

$$v_i - \eta \times \left(\frac{\partial h}{\partial X_i} \right)(v_1, \dots, v_n)$$

η 是步长。偏导是 $\frac{\partial h}{\partial X_i}$ ，它是 X_1, \dots, X_n 的函数。

将它应用到点 (v_1, \dots, v_n) 给出

$$\left(\frac{\partial h}{\partial X_i} \right)(v_1, \dots, v_n) = \lim_{\epsilon \rightarrow 0} \frac{h(v_1, \dots, v_i + \epsilon, \dots, v_n) - h(v_1, \dots, v_i, \dots, v_n)}{\epsilon}$$

如果 h 的偏导可以被解析计算，那么就将它解析。如果不能解析计算，那么就用一个很小的值 ϵ 。

梯度下降法经常被用到参数学习，在参数学习中有可能存在成千上万的实值参数需要被优化。该方法的许多变形这里就不进行介绍了。例如，不用常数的步长，算法用一个二元搜索来决定局部最优的步长。

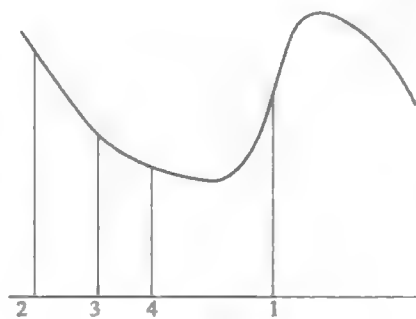


图 4-12 梯度下降

150

4.11 本章小结

- 在 Agent 解决实际问题的过程中，应用描述状态的特征集进行推理的方法要比根据状态显式推理的方法更为有效。
- 许多问题可以用一个变量集合表示，并且存在与该集合对应的特征集、变量可能取值所组成的域，以及硬和/或软约束集合。问题的解是为每个变量赋值的一种赋值方案，该赋值方案被要求满足硬约束集合或者最优化一些函数。

- 弧一致方法和搜索方法经常被结合使用,用来寻找满足约束条件的赋值方案,或者表示问题无解。
- 随机局部搜索用于寻找满足约束条件的赋值方案,但是不能确定问题是否无解。算法在每次改进所消耗的时间和每次移动所改进的值之间做出权衡来保证算法的有效性。这其中还必须要使用一些策略来保证算法能够跳出非解的局部最优值。
- 当约束图是稀疏图时,可以采用系统化的方法进行最优化。这种情况也可以使用局部搜索的方法,但是该方法所带来的问题是无法确认当前的搜索是否正处于全局最优。

4.12 参考文献及进一步阅读

1977年,由Mackworth提出了广义弧一致(GAC)算法。2003年Dechter讨论了约束满足技术,2006年Freuder和Mackworth做了类似工作。

早在1962年,Davis、Logemann和Loveland三人就联合提出了DPLL算法。

针对命题满足问题的变量消除(VE)方法是由Davis和Putnam在1960年提出的。而最优化理论变量消除(VE)可以称为非序列动态规划(non-serial dynamic programming),该方法是由Bertelè和Brioschi在1972年共同提出。

随机局部搜索方法在2003年和2004年分别由Spall和Hoos、Stützle给出相应的描述。Minton、Johnston、Philips和Laird在1992年提出了最小冲突启发式方法,基于该方法衍生出了任意冲突启发式方法。经典的模拟退火算法是由Kirkpatrick、Gelatt和Vecchi在1983年提出的。

遗传算法是由Holland在1975年率先提出。有关遗传算法的文献有许多,相关文献可以查询Goldberg[1989],Koza[1992],Mitchell[1996],Bäck[1996],Whitley[2001]和Goldberg[2002]的相应工作。

4.13 习题

- 4.1 在图4-13的填字游戏中,必须找出6个包含3个字母的单词:3个横向阅读的单词(A1、A2、A3),3个纵向阅读的单词(D1、D2、D3)。必须从右侧显示的40个单词列表中选择每个单词。首先,尝试用直觉来填写;然后用先域一致后弧一致的方法来解决。

至少存在两种方式将图4-13中的填字游戏表达为约束满足问题。

第一种表达方式是将词的位置(A1、A2、A3、D1、D2、D3)作为变量,词的集合作为变量的可能取值。约束条件则为单词交叉位置的字母必须相同。

第二种表达方式是将九个方格作为变量。字母表的字母集合(a, b, ..., z)则为变量的域。约束则为相应位置上的字母必须构成单词表里面的单词。例如,左上方的方格和中间最上方的方格不能同时取a值,因为右侧单词表内没有aa开头的单词。

A1,D1	D2	D3
A2		
A3		

单词表:

add,ado,age,ago,aid,
ail,aim,air,and,any,
ape,apt,arc,are,ark,
arm,art,ash,ask,auk,
awe,awl,aye,bad,bag,
ban,bat,bee,boa,car,
eel,elt,far,fat,fit,
lee,oaf,rat,tar,tie

图4-13 由6个单词完成的填字游戏

- 根据第一种表达,请给出使用域一致方法进行约简的例子(若存在)。
 - 根据第一种表达,请给出使用弧一致方法进行约简的例子(若存在)。
 - 根据第一种表达,使用域一致加弧一致的方法是否足以解决这个问题,请给出相应解释。
 - 根据第二种表达,请给出使用域一致方法进行约简的例子(若存在)。
 - 根据第二种表达,请给出使用弧一致方法进行约简的例子(若存在)。
 - 根据第二种表达,使用域一致加弧一致的方法是否足以解决这个问题,请给出相应解释。
 - 基于一致性技术,哪种表达可以使问题更有效地被解决?在答案中给出证明。
- 4.2 假设存在一个关系 $v(N, W)$, 该关系在单词 W 中的第 N 个字母是元音(a, e, i, o, u 中的一个)时为真。例如, $v(2, cat)$ 为真, 因为 cat 中的第二个字母“a”为元音。 $v(3, cat)$ 为假, 因为 cat 中

的第三个字母“t”不是元音。 $v(5, cat)$ 为假, 因为 *cat* 中不存在第五个字母。

假设 N 的域是 $\{1, 3, 5\}$, W 的域是 $\{added, blue, fever, green, stare\}$ 。

(a) 弧 $\langle N, v \rangle$ 是否为弧一致? 若是, 请给出原因。若不是, 请指出从域中删除哪些元素可以使其形成弧一致。

(b) 弧 $\langle W, v \rangle$ 是否为弧一致? 若是, 请给出原因。若不是, 请指出从域中删除哪些元素可以使其形成弧一致。

4.3 在图 4-14 的填字游戏中, 可以被使用的单词有:

at, eta, be, hat, he, her, it, him, on, one, desk, dance, usage, easy, dove, first, else, loses, fuels, help, baste, given, kind, sense, soon, sound, this, think.

(a) 给出用节点位置(1-横向, 2-纵向等)和单词域的表达, 并具体说明在域一致和弧一致操作后的网络。

(b) 考虑对偶表达方式, 单词的交叉位置为变量, 这些变量的域是能放置在交叉位置的字母。给出在弧一致方法构造网络后的变量域。在这种表达方式下, 使用弧一致方法后得到的结果是否对应了(a)中的网络。

(c) 基于(a)中弧一致操作后得到的网络, 给出使用变量消除法解决该填字游戏的过程。

(d) 不同的消除顺序是否影响效率? 请给出相应解释。

4.4 使用局部随机搜索解决习题 4.3 中的问题。基于弧一致网络, 可以使用“随机局部搜索”Alspace.org 中的小程序来解决问题。

(a) 随机游走的方法效果如何?

(b) 爬山算法的效果如何?

(c) 组合的方法效果如何?

(d) 给出参数集合的设置使得效果最佳。

4.5 一个调度问题中包含 5 个活动, 这 5 个活动需要被安排在 4 个时间段上。假设 A, B, C, D, E 为表示活动的变量, 变量的域为 $\{1, 2, 3, 4\}$, 且约束为 $A > D, D > E, C \neq A, C > E, C \neq D, B \geq A, B \neq C$ 和 $C \neq D + 1$ 。

(先用直觉尝试着给出一种合理的调度安排。)

(a) 给出用回溯法解决该问题的过程。画出生成的搜索树并找到所有解。给出清晰的调度方案, 并且确保选择了一个合理的变量顺序。

为了给出搜索树, 将它以文本的形式写出, 该搜索树的每一个分支占一行。例如, 假设变量 X, Y, Z 的域为 t, f , 且约束 $X \neq Y, Y \neq Z$ 。对应的搜索树可以以如下方式写出:

```
X=t Y=t failure
  Y=f Z=t solution
    Z=f failure
X=f Y=t Z=t failure
  Z=f solution
    Y=f failure
```

(提示: 对于一个具体问题来说, 写一个程序来生成搜索树比手动可能要更容易些。)

(b) 给出用弧一致法解决该问题的过程。必须完成以下操作:

- 画出约束图;
- 给出每步删除域中元素的过程, 以及在删除元素的过程中涉及了哪些弧;
- 在弧一致结束后给出精确的约束图;
- 给出使用分割域方法解决该问题的过程。

4.6 考虑哪些方法可以做到以下的要求:

(a) 如果不存在模型, 能判断出没有模型。

(b) 如果存在一个模型, 能找到这个模型。

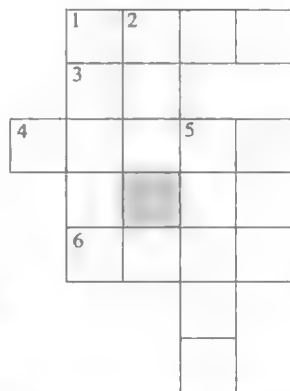


图 4-14 由 7 个单词完成的填字游戏

(c) 确保找到所有模型。

方法如下：

i) 使用域分割的弧一致。

ii) 变量消除。

iii) 随机局部搜索。

iv) 遗传算法。

- 4.7 基于域分割的弧一致方法能返回不仅仅是一个而是所有模型，解释工作流程，并给出算法。
- 4.8 变量消除法能返回一个而不是所有模型，解释工作流程，如何比找到全部模型更容易地找到一个模型，并给出算法。
- 4.9 解释基于域分割的弧一致方法是如何计算模型的数量。
- 4.10 解释基于变量消除的弧一致方法是如何不使用枚举方法而计算模型数量的。(提示：不需要逆向推理，但是可以向前推导解的数量。)
- 4.11 思考图 4-15 中二元约束的约束图(例如， r_1 是 A 和 B 的关系，可以写 $r_1(A, B)$)，用变量消除法解决这个网络。
- (a) 假设要消除变量 A ，哪些约束被删除？在哪些变量上又被创建了一个新约束(可以称之为 r_{11})？
- (b) 假设随后要消除变量 B (例如，在消除 A 之后)，哪些关系将被移除？在哪些变量上又被创建了一个新约束？
- 4.12 将密码算术问题 $SEND + MORE = MONEY$ 作为约束满足问题提出并解决。在密码算术问题中，每个字母都用一个不相同的数字表示，最左边的数字不能为 0(因为这将是不存在的)，且对应的字母序列是一个十进制的数，其和必须是正确的。在本例中， $Y = (D + E) \bmod 10$ ，且 $E = (N + R + ((D + E) \div 10)) \bmod 10$ ，等等。

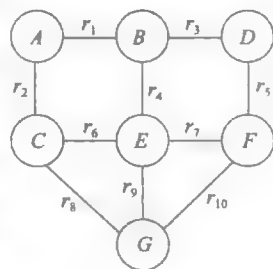


图 4-15 抽象的约束网络

命题和推理

当我遇到一个错误的定理时，我不需要对其进行考察或论证，因为我可以通过一个简单的实验来发现其事实中的错误，即通过对实例的计算，就可以轻而易举地发现其中的错误。

如果有人怀疑我的结论，我就会对他说“让我们来计算一下”，然后付诸于笔端，我们很快就会解决问题。

——Gottfried Wilhelm Leibniz[1677]

本章着重介绍简单形式知识库的相关理论，这种知识库阐述某个环境中的事实。Agent可以观察并使用该知识库的内容，从而确定该环境中的其他事实。在一个给定的知识库中，当 Agent 询问何为真时，知识库不必枚举可能世界就可以回答，甚至可以产生所有的可能世界。本章介绍了使用命题进行的形式化推理。各种形式化推理的不同之处在于证明了什么，什么样的背景知识必须被提供，以及 Agent 在线上获取的观察是怎样被操作的。

5.1 命题

约束作为变量合理赋值的表，具体编写时并不是非常直观，很难看出这些表中所示的内容。而且对知识库进行调试也是一件很困难的事情。一点小的变化都会引起表中内容很大的改变。通常编写约束的方法是以命题的形式进行的。

157

对于使用命题表示约束和查询有如下原因：

- 对于变量间关系的表示使用逻辑声明比使用具体的表达要更简洁、易读。
- 通过这种方式查询 Agent 得到的回答要比对单个变量赋值得到的更为丰富。
- 该语言将在第 12 章中被扩展以对个体和关系进行推理。

首先给出称为**命题演算**(propositional calculus)语言的语法和语义。

5.1.1 命题演算的语法

命题(proposition)是判断真假的语句，在某一个世界中其有一定的真值(如真或者假)。命题是通过原子命题的逻辑连接建立起来的。

原子命题(atomic proposition)，或称为**原子**(atom)，是使用一个小写字母开始的符号表示。直观地说，原子是有真假值的事物。

例如，人工智能是有趣的(*ai_is_fun*)、点亮 l_1 (*lit_l_1*)、住在外边(*live_outside*)和阳光(*sunny*)，都可以称作原子命题。

从前面提到的代数变量的角度看，原子可以看做是有一个特定值或者一组值的声明。例如，命题 *classtimeAfter3* 可以意味着 *ClassTime* > 3，即当变量 *ClassTime* 值大于 3 时，该命题为真，否则为假。传统的命题演算不会将变量显式地描述，本书也遵循这一传统。我们知道，对于**布尔变量**(Boolean variable)存在着一个直接的事实，即命题不为真即为

假。赋值 $X = \text{true}$ 写成命题 x ，这里使用了小写的变量名。因此，命题 *happy* 就意味着存在一个名为 *Happy* 的布尔变量，其中 *happy* 意味着 $\text{Happy} = \text{true}$ 。

命题也可以通过逻辑连接词连接简单命题组成。因此命题是下述两者之一：

- 一个原子命题。
- 一个复合命题(compound proposition)，如下面几种形式：

$\neg p$ (读作非 p)——是 p 的否(negation)

$p \wedge q$ (读作 p 和 q)——是 p 和 q 的合取(conjunction)

$p \vee q$ (读作 p 或 q)——是 p 和 q 的析取(disjunction)

$p \rightarrow q$ (读作 p 蕴含 q)——是从 p 蕴含(implication) q

$p \leftarrow q$ (读作如果 q 那么 p)——是从 q 蕴含 p

$p \leftrightarrow q$ (读作“ p 当且仅当 q ”或“ p 等价于 q ”)

其中 p 和 q 都是命题。

158

复合命题操作符的优先级按照上述顺序排列。因此，我们可以按上面的顺序，使用对复合命题加上括号的方法消除复合命题的歧义，例如：

$$\neg a \vee b \wedge c \rightarrow d \wedge \neg e \vee f$$

是下面命题的缩写

$$((\neg a) \vee (b \wedge c)) \rightarrow ((d \wedge (\neg e)) \vee f)$$

5.1.2 命题演算的语义

语义(semantic)是指语言符号与现实世界事物的对应。可以用语义理解语言中句子的含义。命题演算的语义定义如下：

解释(interpretation)由映射函数 π 组成，映射函数 π 是将原子命题映射到 $\{\text{true}, \text{false}\}$ 的函数。如果 $\pi(a) = \text{true}$ ，则我们说在该解释中原子命题 a 为真，或说该解释将 *true* 赋予 a 。如果 $\pi(a) = \text{false}$ ，我们称在该解释中 a 为假。有时可以认为 π 是一组映射为真的原子，而其他的原子则映射为假。

解释将每个命题映射为一个真值(truth value)。在解释中每个命题可能是真命题也可能是假命题。如果 $\pi(a) = \text{true}$ ，则原子命题 a 在该解释中为真，否则在该解释中为假。复合命题的真值可以使用真值(图 5-1)进行判断。

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \leftarrow q$	$p \rightarrow q$	$p \leftrightarrow q$
<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>
<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>false</i>
<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>false</i>
<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>

图 5-1 定义 \neg 、 \wedge 、 \vee 、 \leftarrow 、 \rightarrow 和 \leftrightarrow 的真值表

注意，我们只谈论在某种解释中的真值。同一命题在不同的解释中可能会有不同的真值。

【例 5-1】 假设有三个原子命题：人工智能是有趣的(*ai_is_fun*)，快乐(*happy*)，点亮灯(*light_on*)。

假设解释 I_1 中：*ai_is_fun* 为真，*happy* 为假，*light_on* 为真。也就是说， I_1 映射的函数 $\pi_1(\text{ai_is_fun}) = \text{true}$ ， $\pi_1(\text{happy}) = \text{false}$ ， $\pi_1(\text{light_on}) = \text{true}$ 。那么

- 在 I_1 中 ai_is_fun 为真。
- 在 I_1 中 $\neg ai_is_fun$ 为假。
- 在 I_1 中 $happy$ 为假。
- 在 I_1 中 $\neg happy$ 为真。
- 在 I_1 中 $ai_is_fun \vee happy$ 为真。
- 在 I_1 中 $ai_is_fun \leftarrow happy$ 为真。
- 在 I_1 中 $happy \leftarrow ai_is_fun$ 为假。
- 在 I_1 中 $ai_is_fun \leftarrow happy \wedge light_on$ 为真。

159

假设解释 I_2 中 ai_is_fun 为假, $happy$ 为真, $light_on$ 为假。

- 在 I_2 中 ai_is_fun 为假。
- 在 I_2 中 $\neg ai_is_fun$ 为真。
- 在 I_2 中 $happy$ 为真。
- 在 I_2 中 $\neg happy$ 为假。
- 在 I_2 中 $ai_is_fun \vee happy$ 为真。
- 在 I_2 中 $ai_is_fun \leftarrow happy$ 为假。
- 在 I_2 中 $ai_is_fun \leftarrow light_on$ 为真。
- 在 I_2 中 $ai_is_fun \leftarrow happy \wedge light_on$ 为真。

知识库(knowledge base)是令 Agent 为真的一组命题的集合, 知识库的元素是公理(axiom)。

一组命题的模型(model)是该组命题都为真的一个解释。

如果 KB 是一个知识库, g 是一个命题, 称 g 是 KB 的一个逻辑结论(logical consequence), 如果 g 在每个 KB 的模型中都为真, 记作

$$KB \models g$$

也就是说, 不存在 KB 为真、 g 为假的解释。若在一个解释中 KB 为假, 那么逻辑结论的定义将不会限制 g 的真值。

如果 $KB \models g$, 我们也说 g 从 KB 逻辑推出(logically follow), 或者说 KB 制约(entail) g 。

【例 5-2】 假设 KB 是下述知识库:

Sam 是快乐的(sam_is_happy)。

人工智能是有趣的(ai_is_fun)。

虫子生活在地下($worms_live_underground$)。

夜晚($night_time$)。

小鸟吃苹果($bird_eats_apple$)。

如果小鸟吃苹果那么苹果被吃($apple_is_eaten \leftarrow bird_eats_apple$)。

如果 Sam 在房间里且是夜晚那么灯是开着的($switch_1_is_up \leftarrow sam_is_in_room \wedge night_time$)。

给定以上的知识库, 则:

$$KB \models bird_eats_apple$$

$$KB \models apple_is_eaten$$

因为存在着使 $switch_1_is_up$ 为假的知识库模型, 所以 KB 不制约 $switch_1_is_up$ 这样的命题。注意在这样的解释中 $sam_is_in_room$ 必定为假。

160

1. 人类角度的语义观

通过之前对语义的描述, 我们并不知道语义的作用, 也不知道如何用语义作为基础信息来构造智能系统。我们知道的是, 使用逻辑背后的基本思想是当知识库的设计者需要实

例化一个特别的世界时,设计者可以选择这个世界作为一种**预期解释**(intended interpretation),可以选择关于这个世界的符号的意义,并且可以写出世界中为真的命题。当系统计算出该知识库的逻辑结果时,设计人员可以根据该预期解释来解释答案。设计人员可以与其他设计人员和用户交流上述的意义,以便于他们也能根据符号的含义解释答案。

逻辑蕴含“ $KB \models g$ ”是一组命题(KB)和它蕴含的命题 g 之间的语义关系。 KB 和 g 都是符号,而且可以在计算机中表示。这些符号可以参照真实世界的命名方式进行命名,但不用考虑真实世界中的语义。 \models 关系是不用计算或证明的,它只是提供了当一些声明为真的时候,接下来会跟随些什么。

知识库设计者表达世界时用到的方法如下所述:

第一步:知识库设计者选择任务域或世界进行表示,这些表示应该是具有预期解释的。这可以是现实世界某个方面的解释(例如一所大学的课程和学生的结构,或在某特定时刻的实验室环境),也可以是某个虚构世界的解释(如爱丽丝梦游的仙境,或一个开关置下后的电气环境),或者是一个抽象世界的解释(例如数和集合的世界)。设计者还必须选择其关注的命题。

第二步:为了表达该世界中的某些方面,知识库设计者需要选择一些原子命题。每个原子命题都关于该预期解释具有确切的意义。

第三步:知识库设计者告诉(tell)系统哪些命题在该预期解释中为真。这经常被称为使领域公理化(axiomatizing),其中给定的命题是领域的公理(axiom)。

第四步:知识库设计者现在可以询问(ask)有关预期解释的问题,系统能回答这些问题。设计人员能够使用赋予符号的意义来解释答案。

使用这种方法,一直到第三步时设计者才实际上告诉计算机一些信息。前两步是在设计者的脑子里进行的。

设计人员应记录符号的含义,以便让其他人理解他们的表示,让人们知道每个符号是什么意思,使他们可以检查给定命题的真实性。符号含义的规范被称为**本体**(ontology)。本体可以在注释中非正式地说明,但是本体越来越多地被用在正式语言中,以便能使语义具有互操作性——互操作性是指在不同知识库中统一使用符号的能力,使得相同的符号具有同一个含义。这部分内容将在本书第13章详细讨论。

第四步可以由其他人来做,只要他们能够理解本体就可以。其他人只需要知道问题和答案中符号的意义,并且相信知识库设计者已经交代的事实为真,就可以推断他们的问题的答案。

2. 计算机角度的语义观

为系统提供信息的知识库设计者享有预期解释,并且根据预期解释来解释符号。设计者对知识库和一组命题进行声明,给出在预期解释中什么是真命题。计算机没有途径使用该预期解释,只有途径使用知识库中的命题。我们会看到,计算机可以辨别某个命题是不是知识库的逻辑结论。如果知识库设计者已经根据符号的意义进行了真值赋值,那么该预期解释就可以看做是公理的模型。假设预期解释是知识库的模型,那么如果某一命题是知识库的逻辑结论,则在该预期解释中该命题为真,因为在知识库的所有模型中它都为真。

逻辑结论的概念看来是从公理化世界推导出隐含信息的有力工具。假设 KB 代表着关于预期解释的知识,也就是说预期解释是知识库的一个模型,是所有系统都知道的预期解释。如果 $KB \models g$,那么 g 在预期解释中一定为真,因为它在知识库的所有模型中都为真。

如果 $KB \not\models g$, 也就是说如果 g 不是该知识库的逻辑结论, 则存在 g 为假的 KB 的模型。对所关心的计算而言, 预期解释可以是其中 g 为假的 KB 的模型, 因而不能确定在预期解释中 g 是否为真。

给定一个知识库且其为真, 则知识库的模型对应着该世界可能被描述的所有形式。

【例 5-3】 考虑例 5-2 的知识库。用户可以解释这些符号的含义, 计算机不知道这些符号的含义, 但是仍然可以根据已被告知信息得出结论。计算机可以得出在预期解释中“苹果被吃”是正确的。计算机因为根据预期解释不知道“Sam 在房间里”是真还是假, 所以不能断定“灯是开着的”这个命题是否正确。

如果知识库的设计者告诉计算机假的信息, 即在预期解释中有一些假的公理, 那么计算机从预期解释中可能无法得出正确的答案。

162

人们需要知道, 如果计算机不具备在世界中感知和行动的能力, 那么计算机就不会知道符号的含义。符号的含义是人为给出的。计算机知道的关于世界的所有东西都是被告知的东西。然而, 由于计算机能够得出知识库的逻辑结论, 所以它可以得到预期解释中为真的结论。

5.2 命题确定子句

命题确定子句(propositional definite clause)语言是命题演算的子语言, 它不允许存在不确定性或歧义性。在这种语言中, 命题与在命题演算中定义的命题具有相同的含义, 但在知识库中不是所有的复合命题都是被允许的。

命题确定子句的语法(syntax)如下:

- **原子命题**(atomic proposition)或**原子**(atom)与在命题演算中的定义是一样的。
- **主体**(body)是一个原子或者多个原子的合取。递归地定义, 一个主体可以是一个原子命题或形如“ $a \wedge b$ ”的形式, 其中 a 是一个原子, b 是一个主体。
- **确定子句**(definite clause)既可以是一个原子 a , 称为**原子子句**(atomic clause), 也可以是形式为 $a \leftarrow b$ 的子句, 称为**规则**(rule), 其中 a 是**头部**(head), 是一个原子, 而 b 是**主体**。
- **知识库**(knowledge base)是一组确定子句的集合。

【例 5-4】 在例 5-2 中知识库的元素全部都是确定子句。

以下不是确定子句:

```
apple_is_eaten  $\wedge$  bird_eats_apple
sam_is_in_room  $\wedge$  night_time  $\leftarrow$  switch_1_is_up
Apple_is_eaten  $\leftarrow$  Bird_eats_apple
happy  $\vee$  sad  $\vee$   $\neg$  alive
```

由于原子必须以小写字母开始, 第三个命题不是确定子句。

需要注意的是, 确定子句是一种受限制的子句形式。例如, 确定子句

$a \leftarrow b \wedge c \wedge d$

等价于子句

$a \vee \neg b \vee \neg c \vee \neg d$

通常来讲, 一个确定子句等价于恰有一个正文字(非否的原子)的子句。命题确定子句不能表示原子的析取(如 $a \vee b$)或原子的非的析取(如 $\neg c \vee \neg d$)。

163

【例 5-5】 考虑如何按照用户角度语义观的方法对图 5-2 的电气图进行公理化。我们

可以使用这种公理化的方法来模拟电力系统。而在后面的章节中，我们会进一步扩展如何基于所观察到的征兆来进行错误诊断。

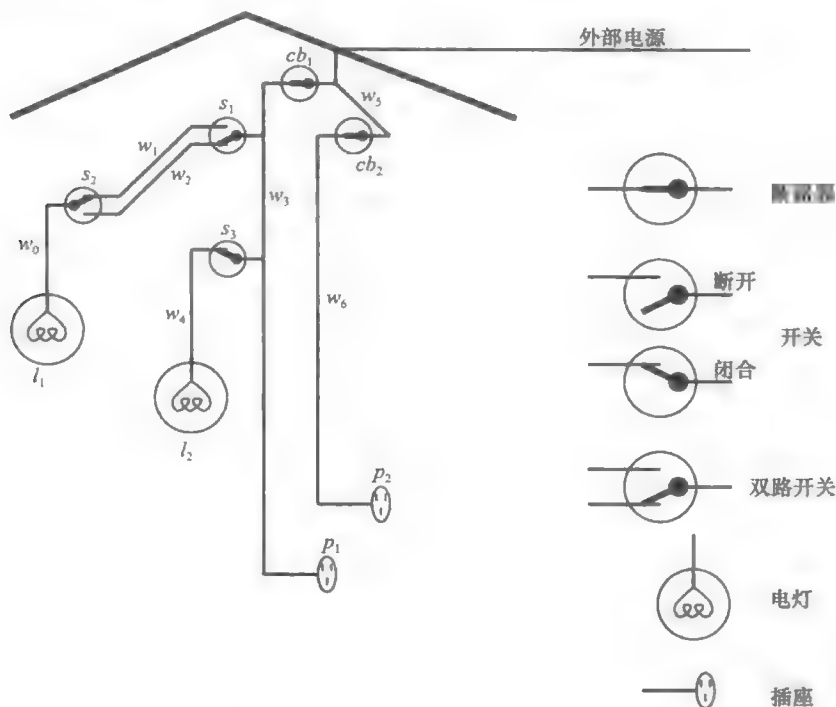


图 5-2 已对组件命名的电气环境

假设根据开关的位置和断路器的状态的表达可以确定灯的状态是开启还是关闭的，如果观察到某些非预料的情况，最终我们也可以判断电线、开关或断路器是否存在问题。当然，前提是我们并不关心电线的颜色、开关的设计、电线的长度或重量、电线和电灯的生产日期，抑或是你可以想象的此领域的其他细节。

接下来，我们选择公理化的抽象层次，其目的是在最通用的层次内对该领域进行表达，通过这种表示，诊断助手可以解决该层次中需要解决的问题。我们同时也希望，这个层次是该环境中 Agent 所能获得的信息层次。例如我们可以表示实际的电压和电流，同样可以表示一个 12 伏直流系统或 120 伏交流系统，但是这对于开关如何影响电灯开闭的问题来说是无关紧要的。相反，我们表示这个领域是在常识层面上，即非电工人员用以描述此领域的层面，就是根据电流通过电线从外面流向开关来点亮灯，并且如果断路器和开关被打开并工作，那么断路器和照明开关是与电线相连的。

我们必须要选择哪些问题是需要表示的。假设我们想表示以下命题：灯是否是亮的，电线是否是通的，开关是开的还是关的，组件是否是坏的。

那么我们要选择该世界中有特殊意义的原子。我们可以用如下描述性的名字，例如用 up_s_2 来表示开关 s_2 是否置上，用 $live_l_1$ 表示灯 l_1 是否是亮的（即是否有电流进入）。计算机并不知道这些名字的含义，也无法了解原子名字的组成。

在这个阶段，我们什么也没有告诉计算机。它不知道什么是原子也不知道这些原子所代表的意思。

一旦我们决定了使用什么样的符号表示什么含义，我们就使用确定子句告诉系统在真

实世界中什么为真的背景知识。定义最简单的确定子句形式如下(这些原子子句没有主体):

$light_l_1$
 $light_l_2$
 ok_l_1
 ok_l_2
 ok_cb_1
 ok_cb_2
 $live_outside$

设计者可以观察到该领域内的部分内容,并且知道如果电线 w_0 是通的那么灯 l_1 就是亮的,因为电线和电灯是连在一起的,但是可能不知道 w_0 是否是通的。表示这样的知识要通过以下规则:

$live_l_1 \leftarrow live_w_0$
 $live_w_0 \leftarrow live_w_1 \wedge up_s_2$
 $live_w_0 \leftarrow live_w_2 \wedge down_s_2$
 $live_w_1 \leftarrow live_w_3 \wedge up_s_1$
 $live_w_2 \leftarrow live_w_3 \wedge down_s_1$
 $live_l_2 \leftarrow live_w_4$
 $live_w_4 \leftarrow live_w_3 \wedge up_s_3$
 $live_p_1 \leftarrow live_w_3$
 $live_w_3 \leftarrow live_w_5 \wedge ok_cb_1$
 $live_p_2 \leftarrow live_w_6$
 $live_w_6 \leftarrow live_w_5 \wedge ok_cb_2$
 $live_w_5 \leftarrow live_outside$
 $lit_l_1 \leftarrow light_l_1 \wedge live_l_1 \wedge ok_l_1$
 $lit_l_2 \leftarrow light_l_2 \wedge live_l_2 \wedge ok_l_2$

在运行时用户能够输入观察到的当前开关的状态,例如

s_1 是置下的($down_s_1$)
 s_2 是置上的(up_s_2)
 s_3 是置上的(up_s_3)

这个知识库由所有上述确定子句组成,不论是背景知识还是观察值,它都进行了表示。

165

5.2.1 问题与解答

为了确定在某特定的世界中还有哪些事物为真,需要为该世界建立描述。给定计算机有关某特定领域的知识库后,用户就可以向计算机询问该领域的有关问题。计算机可以回答该命题是否是该知识库的逻辑结论。如果用户理解该领域中原子的含义,那么用户可以解释该答案。

查询(query)是询问某一命题是否是知识库的逻辑结论的一种方式。一旦系统提供了知识库,就可以使用查询来询问某一公式是否是该知识库的逻辑结论。查询形式如下:

$ask\ b$

其中, b 是主体。

如果主体是知识库的逻辑结论,查询的问题就解答(answer)为“是”,如果不是知识库

的逻辑结论就解答为“否”。后者并不一定意味着在该预期解释中主体是假的，而是基于知识库提供的知识不能确定它是真还是假。

【例 5-6】 如果将例 5-5 的知识库告诉计算机，它可以回答如下的查询。查询

`ask light_L1`

的回答是“是”。查询

`ask light_L6`

的回答是“否”。计算机没有足够的知识知道 L_6 是否是灯。查询

`ask lit_L2`

的回答是“是”，在所有的模型中这个原子都是真的。

用户可以通过预期解释来解释这些答案。

166

5.2.2 验证

到目前为止，我们已经知道了答案是什么，但是不知道它是如何被计算出来的。我们定义了符号 \models 规定哪些命题应是知识库的逻辑结论，但不知道怎样计算它们。通过演绎 (deduction) 我们可以确定某个命题是否是知识库的逻辑结论。演绎是一种特殊形式的推理 (inference)。

验证 (proof) 是一种机械性的可推导过程，表明某一命题从一个知识库逻辑推出。**定理** (theorem) 是可证明的命题。**验证过程** (proof procedure) 是一个用于推导知识库的结论的算法——这个算法可能是不确定的。如下面描述的“不确定性选择”。

给定一个验证过程， $KB \vdash g$ 表示命题 g 可以通过知识库 KB **证明** (proved) 或 **导出** (derived)。

验证过程的质量体现在它是否计算了想要计算的内容。

如果任何可以从知识库中导出的命题都是知识库的逻辑结论，则称该验证过程为**合理的** (sound)。即如果 $KB \vdash g$ ，那么 $KB \models g$ 。

如果对于知识库的每一个逻辑结论都存在一个验证，那么该验证过程称为**完备的** (complete)。即如果 $KB \models g$ ，那么 $KB \vdash g$ 。

我们有两种方式命题确定子句构建验证过程：一种是自底向上的过程，另一种是自顶向下的过程。

1. 自底向上的验证过程

第一个验证过程是**自底向上的验证过程** (bottom-up proof procedure)，即导出逻辑结果。自底向上证明过程如同建房子，我们需要先建好房子结构才能自底向上建房。自底向上的验证过程需要先建立原子命题。该验证方法不同于自顶向下方法，自顶向下方法是起始于一个查询，并试图找到支持该查询的确定子句。有时我们会说自底向上的过程是在确定子句上做**正向链接** (forward chaining)，是从我们所知道的事实去推理，而不是从查询的反向去推理。

自底向上验证的总体思路是基于**推导规则** (rule of derivation)，广义的规则推理形式称为**假言推理**：

如果在知识库中“ $h \leftarrow b_1 \wedge \cdots \wedge b_m$ ”是一个确定子句，而且每个 b_i 都成立，那么 h 也成立。

这个规则中也包含了当 $m=0$ 时的情况；自底向上的验证过程可以在没有主体的确定子句的头部推导出原子命题。

图 5-3 给出了计算确定子句集合 KB 的结论集合 C 的验证过程。在这个验证过程中, 如果 g 是一个原子且 $g \in C$, 程序 DCDeductionBU 运行后有 $KB \vdash g$ 。对于合取, 如果 $g_i \in C$, 其中每个 i 满足 $0 < i \leq k$, 则 $KB \vdash g_1 \wedge \cdots \wedge g_k$ 。

167

```

1: procedure DCDeductionBU(KB)
2:   Inputs
3:     KB: 确定子句集合
4:   Output
5:     原子集合, 即 KB 的逻辑结果
6:   Local
7:     C: 原子集合
8:     C := {}
9:   repeat
10:    select KB 中的 " $h \leftarrow b_1 \wedge \cdots \wedge b_m$ ", 其中, 对所有  $i$ ,  $b_i \in C$ ,  $h \notin C$ 
11:    C := C  $\cup$  {h}
12:  until 没有其他确定子句可以选择
13:  return C

```

图 5-3 计算 KB 结果的自底向上验证程序

【例 5-7】 假设给出系统的知识库 KB

$a \leftarrow b \wedge c$

$b \leftarrow d \wedge e$

$b \leftarrow g \wedge e$

$c \leftarrow e$

d

e

$f \leftarrow a \wedge g$

自底向上的程序中 C 的跟踪值如下:

{}

{d}

{e, d}

{c, e, d}

{b, c, e, d}

{a, b, c, e, d}

在该算法结束时, $C = \{a, b, c, e, d\}$, 因此知识库 $KB \vdash a$, $KB \vdash b$, 等等。

在知识库中从没有使用过最后这条规则。自底向上的验证过程不能推导 f 和 g 。因为可能存在某一模型, 其中的 f 和 g 都为假, 所以 f 和 g 在本验证中不能被推导出来。

如图 5-3 的验证程序可以建立许多命题。

168

合理性(soundness) C 中的每个原子都是知识库 KB 中的逻辑结果。即如果有 $KB \vdash g$ 则 $KB \models g$ 。为了说明这一点, 假设在集合 C 中存在一个不符合知识库 KB 的逻辑结果的原子。如果存在这样的原子, 那么肯定有被首先加入 C 的元素, 该元素在 KB 的所有模型中都不为真。假设这个原子是 h , 并假设其在 KB 的模型 I 中不为真, 那么 h 必是产生的第一个 I 中不为真的元素。由于 h 已经产生, 则在知识库 KB 中必定存在确定子句 $h \leftarrow b_1 \wedge \cdots \wedge b_m$, $b_1 \cdots b_m$ 都是属于集合 C 的(其中存在着一种特殊情况, h 是原子子句, 则 $m=0$)。 b_i 在 h 之前产生且在模型 I 中都为真。在模型 I 中这个子句的头部是假, 主体是真, 因此, 通过确定子句的真值判断出这个子句在 I 中为假。这与 I 是知识库 KB 的模型

这一事实相矛盾。因此,集合 C 中的每个元素都是知识库 KB 的逻辑结果。

复杂性(complexity) 图 5-3 中算法的结束条件和循环次数都是由知识库中确定子句的数量决定的。这很容易看出,因为每个确定子句只能用一次。因此,如果可以索引到确定子句那么内部循环可以连续执行,程序算法的时间复杂度在知识库中是呈线性的。

不动点(fixed point) 图 5-3 算法中最后生成的 C 被称为**不动点**,因为推导规则的任何更进一步应用都不能改变 C 。由于没有再小的不动点了, C 就是**最小的不动点**(minimum fixed point)。

令 I 是最小不动点中每一个原子都为真且每一个不在最小不动点中的原子都为假的解释。为了说明 I 一定是 KB 的一个模型,假设“ $h \leftarrow b_1 \wedge \dots \wedge b_m$ ” $\in KB$ 在模型 I 中为假。这种情况唯一可能发生的途径是,假设 b_1, \dots, b_m 在不动点中,但 h 不在不动点中。通过解释,推导规则可以将 h 加入不动点中,则对于其是否在不动点中产生了一个矛盾。因此,在 KB 中不能存在确定子句解释为假且在不动点中的情况。因此 I 是知识库 KB 的一个模型。

完整性(completeness) 假设 $KB \models g$,则在每个知识库模型中 g 都为真,于是在定义最小不动点的模型 I 中为真,在 C 中也为真,因此 $KB \models g$ 。

上面由最小不动点所定义的模型 I 称为**最小模型**(minimum model),其具有最少的真命题。每一个在 I 中的原子在其他模型中必为真。

2. 自顶向下的验证过程

另一种验证方法是从查询反向搜索或是自顶向下搜索,来确定其是否是给定的确定子句的逻辑结论。这个过程被称为**命题确定子句解析**(propositional definite clause resolution)或**SLD 解析**。其中 SL 代表使用线性策略选择原子, D 代表确定子句。这是一种更为普遍的解析方法。

不确定性选择

在许多人工智能程序中,我们希望从如何求解中分离出来解的定义。通常选用不确定性算法,**不确定性**(non-deterministic)算法指程序中存在没有明确指明的选择。以下是两种不确定性选择的类型:

- **不介意不确定性**(don't-care non-determinism),以图 5-3 中“select”为例。在这种不确定性形式中,若一种选择没有给出解决方案,那么尝试其他替代的选择也是无济于事的。不介意不确定性用在对于有限数量的资源产生大量需求的资源分配问题中。每一次,调度算法需要选出谁得到了哪些资源。结果的正确性并不受选择的影响,但是效率和终止条件可能受其影响。当存在着一个无限序列的选择时,如果最终选择了一个可被重复选择的元素,那么这是一种**公平**(fair)的选择机制。一个元素的问题反复多次不被选中的称为**饥饿**(starvation)。我们要确保任何选择都是公平的。
- **不知道不确定性**(don't-know non-determinism),以后面的图 5-4 中“choose”为例。一个挑选没有得出解决方案并不意味着其他的挑选不会得出解决方案。我们经常讲,在每一点的**预言**(oracle)可以实例化,不同的挑选将会得到不同的解。由于我们的 Agent 无法获知这样的预言,所以 Agent 需要搜索整个空间交替挑选。第 3 章给出了算法搜索空间。

不知道不确定性在计算复杂性理论中扮演着很重要的角色。P 问题就是多项式时间内能够解决的问题。NP 类问题包含了通过预言可以在多项式时间内解决的问题，而预言则是每次选取正确的值，这等价于一个解在多项式时间内可被验证。人们普遍推测 $P \neq NP$ ，这意味着没有这样的预言存在。复杂性理论的最重大发现是在 NP 类里所有的最难问题都是同等复杂的，如果其中一个问题可以用多项式时间来求解，那么所有这些 NP 问题都可以用多项式时间求解。这些问题是 **NP 完全**(NP-complete)的。如果一个问题至少与 NP 完全问题一样难，则这个问题就是 **NP 难**(NP-hard)的问题。

在本书中，我们对于不介意不确定性问题使用“select”术语，对不知道不确定性问题使用“choose”术语。在**不确定性过程**(non-deterministic procedure)中，我们假设每一个预言都可以对应产生合适的筛选。因此，如果在一次筛选中产生挑选状态，则成功，或者如果没有这样的筛选，则会**失败**(fail)。不确定性过程可以有很多答案，其中若有很多筛选则是成功的，若没有合适的筛选则将会失败。我们也明确定义**失败**的筛选为不成功的。预言可以通过搜索来获得。

170

自顶向下的验证过程可以通过答案子句的形式理解。**答案子句**(answer clause)的形式如下：

$$yes \leftarrow a_1 \wedge a_2 \wedge \cdots \wedge a_m$$

其中 yes 是一个特殊的原子。直观上看，当查询的结果是“yes”的时候， yes 就为真。

如果查询为：

$$ask \ q_1 \wedge \cdots \wedge q_m$$

初始答案子句为：

$$yes \leftarrow q_1 \wedge \cdots \wedge q_m$$

给定一个答案子句，自顶向下算法在答案子句主体中挑选一个原子。假设其选择了原子 a_i ，则 a_i 称为**目标**(goal)。算法通过一步一步的**归结**(resolution)来运行。在一步归结中，其选择 KB 中 a_i 作为头部的确定子句。如果没有这样的子句，则归结失败。对于上述答案子句的**分解**(resolvent)选择 a_i 的过程中使用到了确定子句

$$a_i \leftarrow b_1 \wedge \cdots \wedge b_p$$

那么，分解是答案子句：

$$yes \leftarrow a_1 \wedge \cdots \wedge a_{i-1} \wedge b_1 \wedge \cdots \wedge b_p \wedge a_{i+1} \wedge \cdots \wedge a_m$$

也就是说，答案子句中的目标被选择的确定子句的主体进行了替代。

答案(answer)是指有一个空主体($m=0$)的答案子句。因此，这个答案子句即为 $yes \leftarrow$ 。

对于知识库 KB 中查询“ $ask \ q_1 \wedge \cdots \wedge q_k$ ”，**SLD 推导**(SLD derivation)是答案子句的序列 $\gamma_0, \gamma_1, \cdots, \gamma_n$ ，其中

- γ_0 是根据原始查询得到的答案子句，即答案子句 $yes \leftarrow q_1 \wedge \cdots \wedge q_k$ 。
- γ_i 是知识库 KB 中确定子句 γ_{i-1} 的分解。
- γ_n 是答案。

该自顶向下算法的另一种思考方式是搜集所有需要证明的原子的集合 G 。每一个需要被证明的原子是一个**目标**。起始， G 是查询中原子的集合。子句 $a \leftarrow b_1 \wedge \cdots \wedge b_p$ 指目标 b_1, \cdots, b_p 可以取代目标 a 。每个目标 b_i 都是 a 的一个**子目标**(subgoal)。需要被证明的 G 对应了答案子句 $yes \leftarrow G$ 。

171

图 5-4 中的程序是用于解决查询的程序，该程序遵循推导的定义。在这个程序中， G 是答案子句的主体中原子的集合。如果算法中存在一个点，在这个点上必须选择一个确定子句来解决冲突，则这个程序是不确定的。如果有导致 G 为空集的选择，返回 *yes*，否则失败。

此算法将子句主体视为一组原子集合， G 也是一组原子集合，也可以通过将 G 中的重复个体移除来获得这个集合，也可以通过将 G 作为一个原子列表来实现，但这可能存在某个原子出现多次的情况。

注意算法要求每次选择的原子使用相同的选择策略。不需要在不同的选择集合间进行搜索，但是所选集合影响算法的执行效率。一般性的选择策略是在原子的排序中选择最左边的原子。这是一个不公平的策略；当不同的策略失败时，其可能会陷入死循环。最好的选择策略是选择最有可能失败的原子。

【例 5-8】 假设给出知识系统如下：

$a \leftarrow b \wedge c$
 $b \leftarrow d \wedge e$
 $b \leftarrow g \wedge e$
 $c \leftarrow e$
 d
 e
 $f \leftarrow a \wedge g$

询问如下：

ask a

172

下面的推导是对应于图 5-4 的重复循环中对于 G 的循序分配。假设 G 通过有序表表示，并且总是选择 G 中最左边的原子：

$yes \leftarrow a$
 $yes \leftarrow b \wedge c$
 $yes \leftarrow d \wedge e \wedge c$
 $yes \leftarrow e \wedge c$
 $yes \leftarrow c$
 $yes \leftarrow e$
 $yes \leftarrow$

下面是选择序列，当选择了对应于 b 的第二个确定子句时，证明过程失败。

$yes \leftarrow a$
 $yes \leftarrow b \wedge c$
 $yes \leftarrow g \wedge e \wedge c$

如果选择了 g ，则没有其他规则可以选择，那么证明的尝试失败。

```

1: non-deterministic procedure DCDeductionTD(KB, Query)
2:   Inputs
3:     KB: 确定子句集合
4:     Query: 待证明原子集合
5:   Output
6:     yes 如果  $KB \models Query$ ，否则程序失败
7:   Local
8:      $G$  为原子集合
9:      $G := Query$ 
10:  repeat
11:    select  $G$  中的一个原子  $a$ 
12:    choose  $KB$  中以  $a$  为头部的确定子句 " $a \leftarrow B$ "
13:    将  $G$  中的  $a$  替换为  $B$ 
14:  until  $G = \{\}$ 
15:  return yes

```

图 5-4 自顶向下确定子句验证程序

当自顶向下的程序已经得出答案后,可以使用推导中的规则在自底向上程序中对查询进行推导。同样,自底向上的原子证明也可以使用自顶向下的推导过程。这种等效可以用于表明自顶向下的证明过程的合理性和完整性。按照定义,自顶向下的证明过程多次重复证明相同的原子花费额外的时间,而自底向上的程序每个原子只证明一次。然而,自底向上的程序证明每一个原子,而自顶向下的程序证明与查询相关的原子。

图 5-4 的自顶向下的非确定性算法结合选择策略就引出了树形搜索图,搜索图中每个节点表示一个答案子句。邻域节点子句 $yes \leftarrow a_i \wedge \dots \wedge a_m$, 其中 a_i 是选择的原子,通过确定 a_i 来找到所有可能的答案子句。对于每一个头部为 a_i 的子句都有一个邻居节点。每一次搜索的目标节点的形式为: $yes \leftarrow$ 。

【例 5-9】 给定如下的知识库:

$a \leftarrow b \wedge c$	$a \leftarrow g$	$a \leftarrow h$
$b \leftarrow j$	$b \leftarrow k$	$d \leftarrow m$
$d \leftarrow p$	$f \leftarrow m$	$f \leftarrow p$
$g \leftarrow m$	$g \leftarrow f$	$k \leftarrow m$
$h \leftarrow m$	p	

询问如下:

ask $a \wedge d$

在 SLD 推导的搜索图中,假设在每个答案子句中中选择最左边的原子,搜索图如图 5-5 所示。

对于大多数问题,我们都没有给出静态的搜索图,因为这需要预测每一个可能的查询。更为现实的是动态构造搜索图。所需要的方法是生成节点的邻居。所选择的答案子句中的原子决定了邻居集合。对于每一个所选择的原子作为头部的规则都存在一个邻居。

第 3 章中的所有搜索方法都可用于进行搜索空间的搜索。因为我们只关心查询结果是否是合乎逻辑的,所以我们只是需要一个目标节点的路径而不是一条最佳路径。每个查询有不同的搜索空间。在每一步选择不同的需要解决的原子会导致搜索空间的不同。

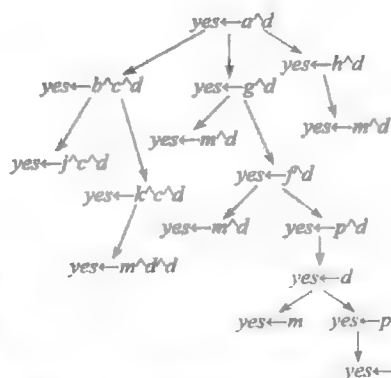


图 5-5 自顶向下推导的搜索图

5.3 知识表示问题

5.3.1 背景知识与观察

观察(observation)是一种在线接收的来自用户、传感器或其他知识源的信息。本章假设一个观察是一个原子命题,由于不同观察是隐式连接的,所以一组观察可表示为一组连接的原子命题。虽然用户和传感器都无法从观察到的世界中直接获取规则,但是 Agent 可通过背景知识从观察中得出有用的内容。

在许多推理框架中,观察补充到背景知识中。但在其他推理框架中(例如,溯因推理、概率推理、学习等),观察与背景知识相分离。

不能期望用户提供的内容都是真实的。首先,用户不确定知识的相关性,其次,他们也不确定采取何种词汇来表达。本体(ontology)和用户界面可解决词汇使用的问题,前者指明符号含义,后者允许用户选取真实含义。然而,许多问题太过复杂,如:知识相关性取决于其他知识的真实性,而繁复的相关事实,使得用户即使通过复杂的图形用户界面也难以确定所有内容的真实性。

同样,被动传感器可提供原子命题合取的直接观察,但主动传感器必须接收 Agent 发出的任务所需的查询信息。

5.3.2 询问用户

在设计或离线时,通常难以获得特例信息。特例信息源于在线的用户、传感器等外部知识源。例如,医疗诊断程序将可能的疾病和症状表示成确定子句,但它不具有特定病人的实际症状等信息。因为用户通常不确定哪些信息相关,也不了解表示语言的语法,所以不能期望用户来提供一个特例的所有信息。作为用户,更愿意通过自然语言来回答实际问题。**询问用户**(querying the user)的思路是向用户求证相关问题,使得用户成为系统的信息源,从而确定哪些信息是相关的,并有助于验证查询。

在自顶向下的验证程序中融入询问用户机制是从用户获取信息的最简单的方法。在该机制下,如果用户在运行时知道某原子的真值,则该原子是**可询问的**(askable)。自顶向下的验证过程中,待验证的原子除可使用知识库中的命题子句来证明外,当原子可询问时,还可询问用户该原子是否为真,而用户只被询问与查询相关的原子。可以选择的原子分为以下三种类型:

- 对于用户未知答案的原子,系统无需询问。
- 对于用户还没有提供答案的可询问原子,可以询问用户,并记录答案。
- 对于用户已经提供了答案的可询问原子,使用该答案,无需询问用户。

自底向上的验证过程也可使用询问用户机制,但应避免询问所有的可询问原子,见习题 5.5。

需注意,系统角色和用户角色具有对称性,均可提出问题也可回答问题。从顶层的角度,用户询问系统问题,而在每一步中,系统询问问题,并通过寻找相关确定子句或者询问用户获得回答。整个系统可以用问题和答案的一套规约来描述。

【例 5-10】 在例 5-5 的电气领域中,无需期望房子的设计师了解开关的位置或者期望用户知道哪个开关连接到哪条电线上。合理的做法是,除开关位置外,所有确定子句均由设计师给出,然后将开关位置设置成为可被询问的。

下面是可能的对话,由用户提出查询,并根据询问回答是或否。此时,用户界面仅是实现基本思路的简单界面,真实的系统应提供更为精细和友好的用户界面。

日志: ask *lit*₁。

*up*_{s₁} 为真吗? [是, 否, 未知, 为什么, 帮助]: 否。

*down*_{s₁} 为真吗? [是, 否, 未知, 为什么, 帮助]: 是。

*down*_{s₂} 为真吗? [是, 否, 未知, 为什么, 帮助]: 是。

回答: *lit*₁。

系统只询问用户可以回答和与手头任务相关的问题。

比起回答问题,更好的情况是用户所指出的异常的事情。比如说,病人可能无法确定所有问题是否为真,但可指出哪些是异常的。例如,一个病人进门后可能会说他的左膝盖疼,那么希望用户主动说他们左肘或者身体其他器官不疼是一种不合理的要求。虽然传感器可能无法识别出该场景中的事物,但是可能识别场景中的变化。

当一个用户指出每件事都存在异常时,Agent 通常可以从欠缺的知识中推断出一些情况。此时,常态可被重载为以异常信息为内容的**默认设置**(default),这种允许默认和异常作为默认设置的思路将在 5.5 节详细论述。

5.3.3 知识层的解释

使用者可以使用语义在知识层(knowledge level)上解释和调试。为确保系统可为人用,系统不能仅给出答案或期望用户相信它。假设存在一个为医生提供建议的系统,医生根据诊断进行处置。医生必须确信诊断是适当的。而系统必须能够验证它的答案是正确的。我们可以使用相同的特征来解释系统是如何获得结果以及调试知识库的。

176

可以使用三种互补的疑问句来解释相关的知识:1)如何做到的问题——“how”用来解释如何得出某一答案;2)为什么的问题——“why”用来问系统为什么问用户这个问题;3)为什么不的问题——“whynot”用来问某一原子不能被证明的原因。

为了解释答案是如何得出的,当系统返回答案时,用户可以询问“如何做到”的 how 问题,系统提供用于推断答案的确定子句。对于确定子句主体中的任何原子,用户可以向系统询问是如何导出该原子的。

用户可以通过问“为什么”的 why 问题来回应被问的问题。该系统将推导问题的规则作为回复。然后,用户可以继续问为什么该规则的头部原子可被证明。用户通过系统回复的这些规则遍历验证过程或顶级查询的部分验证。

“为什么不”的 whynot 问题可以用来问某一原子不能被证明的原因。

1. 系统如何来证明某一目标

第一个解释程序允许用户询问“如何产生”的 how 问题,即“目标是如何导出的?”。如果 g 可被证明,则 g 为一个原子子句或者存在规则:

$$g \leftarrow a_1 \wedge \dots \wedge a_k$$

使得每一个 a_i 都可被证明。

如果系统已经导出 g ,同时,用户再询问“how”,系统可以显示证明 g 的子句。如果子句是一个规则,用户可继续询问:

how i

系统将会给出证明 a_i 的规则。用户可以继续使用 how 问题来探讨 g 是如何导出的。

【例 5-11】在例 5-5 公理化的例子中,用户询问 ask lit_l_2 。为了响应系统对该查询的证明,用户可以问 how 问题。该系统将回复:

$lit_l_2 \leftarrow$

$light_l_2 \wedge$

$live_l_2 \wedge$

ok_l_2

以上为证明 lit_l_2 的顶层规则。为探究如何证明 $live_l_2$,用户可以问:

how 2

177

系统返回用来证明 $live_l_2$ 的规则,即

$live_l_2 \leftarrow$

$live_w_4$

为探究如何证明 $live_w_4$,用户可以问:

how 1

系统呈现规则如下:

$live_w_4 \leftarrow$

$live_w_3 \wedge$

up_s_3

为了探究如何证明子句主体中的第一个原子，用户可以问：

how 1

第一个原子 $live_w_3$ 可通过如下规则证明：

$live_w_3 \leftarrow$
 $live_w_3 \wedge$
 ok_cb_1

为了探究如何证明子句主体中的第 2 个原子，用户可以问：

how 2

系统会报告 ok_cb_1 已被显式地给出。

请注意这里的解释仅从知识层给出，系统仅回复被告知的确定子句。用户不必知道证明过程或者是实际的计算过程。

2. 系统问问题的原因

另一种有用的解释是某一问题会被询问的原因。理解这个解释是非常有用的，因为：

- 我们希望系统是智能的，那么了解问题被问的原因可提高用户对系统智能运行的信心。
- 交互式系统复杂性的主要测量是通过询问用户问题的数量决定的，而如果想最小化系统的复杂性，那么明确问题被问原因能够帮助知识库设计者降低设计的复杂性。
- 一个不相关的问题通常是更深层次问题的征兆。
- 通过了解系统做事的原因，可使用户从系统中学习到知识。这种学习类似于新手问熟手做事的原因。

当系统询问用户一个问题(q)时，系统必须有一个主体中包含了 q 的命题规则。用户可以问：

why

这就好像说：“你为什么问我这个问题呢？”答案是主体中包含 q 的简单规则。如果用户再次询问，系统应该解释为什么位于规则头部会被询问，等等。继续询问为什么会给出到达顶级查询的目标的路径。如果给出的所有规则均是合理的，则证明系统对用户的问题是合理的。

【例 5-12】 考虑例 5-10 的对话。本例中对话表明如何反复使用 why 问题来寻找顶级目标。下面的对话用来询问 lit_l_1 ，由用户提出查询，并回应 why：

日志:ask lit_l_1
 up_s_1 为真吗？为什么？
 up_s_1 被用于规则 $live_w_1 \leftarrow live_w_3 \wedge up_s_1$ ：为什么？
 $live_w_1$ 被用于规则 $live_w_0 \leftarrow live_w_1 \wedge up_s_2$ ：为什么？
 $live_w_0$ 被用于规则 $live_l_1 \leftarrow live_w_0$ ：为什么？
 $live_l_1$ 被用于规则 $lit_l_1 \leftarrow light_l_1 \wedge live_l_1 \wedge ok_l_1$ ：为什么？

因为这是你问我的问题！

通常，how 问题和 why 问题一起使用，how 问题从高目标移向低目标，why 问题从低目标移向高目标。用户同时使用这两类问题可遍历整个证明树(proof tree)。在证明树中，节点是原子，每个节点及其子节点对应知识库中的一个命题子句。

【例 5-13】 考虑先前用户询问 why up_s_1 来给出一个组合了 how 问题和 why 问题的

例子。系统提供如下规则：

$$live_w_1 \leftarrow live_w_3 \wedge up_s_1$$

这就意味着，问 up_s_1 是因为系统想要知道 $live_w_1$ 并且用这个规则试着证明 up_s_1 。用户虽然认为系统想知道 $live_w_1$ 是合理的，但却认为询问 up_s_1 是不合适的，原因在于用户可能质疑 $live_w_3$ 是否会成功。在本例中，用户询问 $live_w_3$ 的导出原因是非常有用的。

5.3.4 知识层的调试

正如在其他软件中一样，知识库可能会有错误或者漏洞。领域专家和知识工程师必须能够调试知识库并添加知识。在基于知识的系统中，由于领域专家和具有领域知识的用户不一定知道或不想知道系统内部是如何工作的，所以调试变得很困难。标准调试工具，如基于执行的轨迹等，均是无效的，因为它们需要首先知道产生答案的知识机制。本节将说明如何使用语义的思路为知识库提供强有力的调试工具，即无论是谁来调试系统都仅需知道符号的含义和特定原子的真假，而这类知识恰好是知识领域专家和用户所具备的。

179

知识层调试(knowledge-level debugging)指仅参考符号的意义而发现知识库中的错误的一种行为。建立可被领域专家使用的知识库系统的目标之一是知识库应该准确地符合该知识领域。例如，对于一个医学知识库的调试，应由医学领域而非人工智能专家完成。同样，对于室内布线知识库的调试，应参考特定的房子，而非基于知识库的内部系统推理。

基于规则的系统中可能出现 4 种类型的非语法错误：

- 产生不正确的答案，即在预期的解释中导出的原子为假。
- 某些答案没有产生，即本应成功证明却被证明失败(没有得出一些特定的真原子命题)。
- 程序陷入死循环。
- 系统询问不相关的问题。

以下是调试前三种错误的检查方法，系统询问不相关的问题可以通过前述的 why 问题进行检查。

1. 不正确答案

不正确答案(incorrect answer)是指在预期解释中已经被证明为假的答案，也叫**错肯定错误**(false-positive error)。如果使用不正确的确定子句，则会在合理性证明中产生不正确答案。

假设无论是领域专家还是用户调试知识库时，均知道语言中符号的预期解释，并可以判断预期解释中某一特定命题是真或假。为了调试不正确答案，领域专家无需知道生成的答案，只需要对问题回答是或否即可。

假设有一个在预期解释中已被证明为假的原子 g 。那么在知识库中必有一个用于证明 g 的规则 $g \leftarrow a_1 \wedge \cdots \wedge a_k$ ，使得下面两种情况中至少有一个成立：

- 在预期解释中 a_i 中有一个为假，在这种情况下， g 可以通过同样的方式进行调试。
- 在预期解释中 a_i 中所有原子为真。在这种情况下，确定子句 $g \leftarrow a_1 \wedge \cdots \wedge a_k$ 一定是不正确的。

180

基于上述情况给出图 5-6 中的算法, 该算法在预期解释所导出原子命题为假时, 可用于调试知识库。算法仅需调试人员回答是或否的问题即可。

该过程也可通过用户提出“how”问题来进行。给定一个在预期解释中为假的原子 g 的证明, 用户可询问该原子是如何被证明的。知识库将返回用于证明 g 的确定子句。如果子句是一条规则, 用户可以询问规则主体中在预期解释中为假的原子“how”问题, 这将返回证明该原子的规则。用户可以重复此操作, 直到子句主体中的所有元素均为真(或者元素为空), 可判断这个子句为不正确的确定子句。上述调试方法假设用户只需确定预期解释中原子是否为真, 而无需知道所使用的证明程序。

【例 5-14】考虑例 5-5 提到的电气领域, 该程序存在一个漏洞。假设该领域专家或用户无意中, w_1 与 w_3 是否连接取决于 s_3 的状态, 而不是 s_1 的状态(见图 1-8)。因此, 知识包括以下不正确的规则:

$$live_w_1 \leftarrow live_w_3 \wedge up_s_3$$

其他公理与例 5-5 相同。给定公理集, 可以导出在预期解释中为假的原子 lit_l_1 。考虑用户检测到该不正确的答案时, 如何发现与之相关不正确的确定子句?

在给定的预期解释中, lit_l_1 为假, 用户询问如何得出, 系统给出规则:

$$lit_l_1 \leftarrow light_l_1 \wedge live_l_1 \wedge ok_l_1$$

再检查该规则主体内的原子。在预期解释中 $light_l_1$ 和 ok_l_1 为真, 但 $live_l_1$ 为假。所以继续询问:

how 2

则系统展示规则:

$$live_l_1 \leftarrow live_w_0$$

在预期解释中 $live_w_0$ 为假, 因此继续询问:

how 1

系统展示规则:

$$live_w_0 \leftarrow live_w_1 \wedge up_s_2$$

在预期解释中 $live_w_1$ 为假, 因此继续询问:

how 1

系统展示规则:

$$live_w_1 \leftarrow live_w_3 \wedge up_s_3$$

在预期解释中这两种元素都为真, 所以这个规则是漏洞规则。

用户或领域专家仅需要知道知识库的预期解释和使用“how”问句的规则, 就可找到存在漏洞的确定子句, 而不必知道系统内部运作方式或证明过程。

2. 缺失的答案

当没有产生预期的答案时, 就产生了第二种类型的错误, 即本应产生答案, 但没有产生该答案时就是缺失的答案。在一个领域中, 虽然目标 g 为真, 但是根据知识库的推理却

```

1: procedure Debug( $g, KB$ )
2:   Inputs
3:      $KB$  是一个知识库
4:      $g$  为一个原子;  $KB \vdash g$  而且  $g$  在预期解释中为假
5:   Output
6:      $KB$  中的不正确子句
7:     找到证明  $g$  的确定子句:  $g \leftarrow a_1 \wedge \dots \wedge a_k \in KB$ 
8:   for each  $a_i$  do:
9:     询问用户  $a_i$  是否为真
10:    if 用户指出  $a_i$  为假 then
11:      返回 Debug( $a_i, KB$ )
12:   return  $g \leftarrow a_1 \wedge \dots \wedge a_k$ 

```

图 5-6 调试不正确答案的算法

没有产生真的结果, 被称为**错否定错误**(false-negative error)。

在这种情况下, 由于没有对 g 的证明, 前面的算法不起作用。而我们必须找出 g 未被证明的原因。

仅当知识库不存在某些确定子句时, 不会产生相应的答案。由于领域专家知道预期解释中的符号含义以及哪些查询会成功(即在预期解释中哪些为真), 因此他们可以对缺失的答案进行调试。给定某一本应被证明的原子命题, 图 5-7 展示了找到缺失的确定子句的算法。

```

1: procedure DebugMissing( $g, KB$ )
2:   Inputs
3:      $KB$  为知识库
4:      $g$  为一个原子:  $KB \not\models g$  且  $g$  在预期解释中为真
5:   Output
6:     缺失的子句中的原子
7:   if 存在确定子句  $g \leftarrow a_1 \wedge \dots \wedge a_n \in KB$  使得所有的  $a_i$  在预期解释中都为真, then
8:     select 不能被证明的  $a_i$ 
9:     DebugMissing( $a_i, KB$ )
10:  else
11:    return  $g$ 

```

图 5-7 缺失答案的调试算法

假设 g 是一个本应被证明但实际上却没有被证明的原子。由于 g 未被证明, 则以 g 为头部的所有确定子句的主体都将失败。

- 假设有关 g 的确定子句集合中存在一条子句应该被证明, 但是却没得到证明; 这意味着在子句主体中所有的原子在预期解释中都必须为真。由于子句主体证明失败, 子句主体中必有一原子证明失败, 但该原子在预期解释中为真, 故可使用递归的方式进行调试。
- 否则, 不存在证明 g 的确定子句, 则用户必须为 g 添加一个确定子句。

【例 5-15】 在例 5-5 的公理化电气领域中, 假设图 1-8 的 s_2 是关闭的。由于缺少 s_2 关闭的确定子句, 例 5-5 的公理化中 lit_l_1 本应被成功证明但却证明失败。下面来找到其中的漏洞。

lit_l_1 失败, 找到所有头部中含有 lit_l_1 的规则, 其中一个规则如下:

$lit_l_1 \leftarrow light_l_1 \wedge live_l_1 \wedge ok_l_1$

用户验证命题中所有的元素都为真。主要表现为, $light_l_1$ 和 ok_l_1 都可以被导出并验证, 但 $live_l_1$ 不可以, 需要对其进行调试。考虑以 $live_l_1$ 为头部的一个规则:

$live_l_1 \leftarrow live_w_0$

原子 $live_w_0$ 不能被证明, 但用户在预期解释中认定其为真, 故可以发现 $live_w_0$ 的规则:

$live_w_0 \leftarrow live_w_1 \wedge up_s_2$

$live_w_0 \leftarrow live_w_2 \wedge down_s_2$

用户发现, 第二条规则的主体为真。主要表现为, 对于 $live_w_2$ 的证明已存在, 但是缺乏对于 $down_s_2$ 的确定子句, 因此返回该原子。修正方法是为该原子添加适当的原子命题或规则。

3. 死循环

如果证明中有一个原子 a 以自身作为子目标, 那么在自顶向下的推导中会出现死循环

(假设子目标是可传递的, 一个子目标的子目标仍是其子目标)。因此, 当知识库有环时, 就会存在死循环。如果存在原子使得确定子句的序列为如下的形式, 则称知识库是有环(cyclic)的:

$$\begin{aligned} a &\leftarrow \dots a_1 \dots \\ a_1 &\leftarrow \dots a_2 \dots \\ &\dots \\ a_n &\leftarrow \dots a \dots \end{aligned}$$

(如果 $n=0$, 则存在一个确定子句, 其头部和主体都为 a 。)

将原子赋值成自然数(非负整数), 如果确定子句主体中的所有原子都被标记成比其头部中的原子更小的数, 则知识库是无环(acyclic)的。本章前面提到的知识库都是无环的。在无环的知识库中不存在死循环。

为了检测知识库中是否有环, 可以修改自顶向下的证明程序, 对每一个原子的所有父原子(ancestor)集进行标记。在图 5-4 的程序中, 集合 A 包括了原子和其父原子集。

将每一个原子的父原子集都初始化为空。当使用下面的规则

$$a \leftarrow a_1 \wedge \dots \wedge a_k$$

来证明 a 时, a_i 的父原子集为 a 和 a 的父原子集的并集, 即

$$\text{ancestors}(a_i) = \text{ancestors}(a) \cup \{a\}$$

如果一个原子在其父原子集中, 则说明知识库有环, 证明失败。值得注意的是, 由于每一个原子都有自己的父原子集, 因此, 这种循环检查相对于之前的版本更加精简。

有环知识库通常是存在漏洞的一个标志。因此在编写知识库时, 通过每次迭代所减少的数值来确定知识库中无环的方法是非常有用的。例如, 在电气领域中, 距离房子外面的步数在每次循环中是减少的。规则且显式地使用熟知排序可以用来预防死循环, 正如在传统语言的编程中必须小心编码才能防止死循环一样。

184

注意, 自底向上的验证过程不会陷入死循环, 因为其只选择头部没有被导出的规则。

5.4 反证法验证

如果允许规则给出矛盾, 则可在反证法证明中使用确定子句。例如, 在电气布线领域中, 指出某些预测非真是很有用的, 例如 *light_l2 is on* 非真。该方法使得诊断推导可以推理出哪些开关、灯或者断路器是损坏的。

5.4.1 Horn 子句

确定子句不允许存在矛盾表述。然而, 我们可以使用反证法来验证语句的简单扩展。

完整性约束(integrity constraint)是如下形式的子句:

$$\text{false} \leftarrow a_1 \wedge \dots \wedge a_k$$

其中 a_i 是原子, *false* 是一个在所有的解释中都为假的特殊原子。

Horn 子句(Horn clause)既可以是一个确定子句, 也可以是一个完整性约束子句。这意味着, Horn 子句头部可以为 *false* 或者是一个一般原子。

在一个知识库的所有模型中完整性约束允许系统验证某些原子的合取为假, 即验证原子否定的析取为真。回想 $\neg p$ 是 p 的否定(negation), 当 p 在某个解释中为假的时候 $\neg p$ 为真, 而且 $p \vee q$ 是 p 与 q 的析取(disjunction), 当 p 为真或 q 为真或二者都为真时, $p \vee q$

为真。完整性约束 $false \leftarrow a_1 \wedge \cdots \wedge a_k$ 逻辑上等价于 $\neg a_1 \vee \cdots \vee \neg a_k$ 。

Horn 子句知识库暗含原子的否定, 如例 5-16 所示。

【例 5-16】 考虑知识库 KB_1 :

$false \leftarrow a \wedge b$

$a \leftarrow c$

$b \leftarrow c$

在 KB_1 的所有模型中原子 c 都为假。如果 c 在 KB_1 的模型 I 中为真, 那么 a 和 b 在 I 中都将是真(否则 I 就不是 KB_1 的一个模型)。因为在 I 中 $false$ 是假的, 而 a 和 b 在 I 中是真的, 在 I 中的第一个子句就是假的。这与 I 为 KB_1 的一个模型矛盾。因而 c 在 KB_1 的所有模型中都是假的。

这个可以表示为:

$KB_1 \models \neg c$

这意味着, 在 KB_1 所有的模型中 $\neg c$ 都为真, 所以 c 在 KB_1 的所有模型中都为假。◀

尽管 Horn 子句语言中不允许将析取和否定作为输入, 但是原子否定的析取可以被推导出来, 如例 5-17 所示。

【例 5-17】 考虑知识库 KB_2 :

$false \leftarrow a \wedge b$

$a \leftarrow c$

$b \leftarrow d$

$b \leftarrow e$

在 KB_2 的每一个模型中, c 或 d 之一为假。如果它们在 KB_2 的模型 I 中都是真的, 那么 a 和 b 在 I 中也将是真的。结果就是 I 中的第一个子句是假的。这与 I 是 KB_2 的模型矛盾。相似的, c 或 e 之一在 KB_2 中的每一个模型都将是假的。因而

$KB_2 \models \neg c \vee \neg d$

$KB_2 \models \neg c \vee \neg e$ ▶

如果子句集合没有模型, 则其为不可满足的(unsatisfiable)。如果可以通过验证程序从子句中推导出 $false$, 那么子句集合则被证明是不一致的(inconsistent)。如果验证程序是合理并且完整的, 则当且仅当子句集合是不可满足的情况下, 子句集合是不一致的。

通常能够为确定子句集合找到一个模型。对于任何一个确定子句集合, 所有的原子都为真的解释就是其一个模型。因此, 确定子句知识库总是可满足的。然而, Horn 子句的集合可能是不可满足的。

【例 5-18】 子句集合 $\{a, false \leftarrow a\}$ 是不可满足的。因而没有解释能够使两个子句同时满足。所以 a 和 $false \leftarrow a$ 在任意解释中都不能同时为真。▶

通过使用 $false$ 作为查询, 自顶向下和自底向上的过程都能够证明不一致性。这就意味着, 当且仅当 $false$ 可以被推导出来时, Horn 子句才是一致的。

5.4.2 假说与冲突

从矛盾中进行推理是一种非常有效的方法。对于一些行为, 知道某些假设的组合是不一致的这很奏效。例如, 得知计划中 Agent 想要完成的行动的组合是不可实现的。在设计中知道一些组件的组合无法协同工作也是有实用价值的。

在诊断应用中, 通过对系统的观察, 可以证明某些组件正常工作时存在矛盾。假设我

们考虑某一个系统, 我们知道其应当如何工作, 而且我们也有一些对该系统工作的观察结果。如果系统没有按照其规范进行工作, 则诊断 Agent 必须确定是哪个组件出现了错误。

为了完成这项任务, 做出可被证明为假的假设是很有必要的。

假说(assumable)是在反证法验证过程中可以被假设的原子。反证法验证可以从假说否定的析取得到。

通过 Horn 子句知识库和明确的假说, 如果系统可以从假设中验证矛盾的存在, 那么系统能提取到不可以全部为真的假设组合。

在确定子句知识库中, 可以使用查询来询问某命题是否是该知识库的结果。给定一个查询, 系统可对该查询构建验证。系统试图通过反证法验证 *false*。作为查询答案的一部分, 用户必须指出什么是被允许的。

如果 KB 是 Horn 子句的集合, KB 的冲突(conflict)是一个假说集合, 该集合对给定 KB 蕴含 *false*。即当以下条件成立时, 称 $C = \{c_1, \dots, c_r\}$ 是对于 KB 的一个冲突:

$$KB \cup \{c_1, \dots, c_r\} \models false$$

在这种情况下, 答案就为:

$$KB \models \neg c_1 \vee \dots \vee \neg c_r$$

最小冲突(minimal conflict)是无严格子集也为冲突的冲突。

【例 5-19】 在例 5-17 中, 如果 $\{c, d, e, f, g, h\}$ 是一个假说集合, 那么 $\{c, d\}$ 和 $\{c, e\}$ 是对于 KB_2 的最小冲突; $\{c, d, e, h\}$ 也是一个冲突, 但不是最小冲突。◀

在接下来的例子中, 通过假说关键字来指明假说, 这些关键字是一个或用逗号间隔的多个假说原子。

5.4.3 基于一致性的诊断

对哪些在正常工作做出假设, 推导出哪些组件工作异常, 这是**基于一致性的诊断**(consistency-based diagnosis)的基础。假设**缺陷**(fault)是指在系统中出现了错误, 则通过系统的模型和观察基于一致性的诊断可判断出可能存在的缺陷。通过不存在缺陷的假说, 可以使用冲突来验证系统中存在的错误。

【例 5-20】 考虑图 5-2 绘制、例 5-5 表述的房屋布线的例子。适用于基于一致性诊断的背景知识库在图 5-8 中给出。在子句中加入一般性假设, 使组件工作的假设变得明确。并且增加以下条件: 开关、电路中断器和灯都如预期一样正常工作。没有对应 *ok* 原子的子句, 但是可以做出这样的假说。

用户可以观察开关的状态和灯是明或暗。

一盏灯不可能同时是明和暗两种状态。该知识按下列完整性约束表示:

$$false \leftarrow dark_l_1 \wedge lit_l_1$$

$$false \leftarrow dark_l_2 \wedge lit_l_2$$

假设用户观察到所有三个开关都置上, 而且 l_1 和 l_2 都是暗的。那么用以下的原子语句代表:

$$up_s_1$$

$$up_s_2$$

$$up_s_3$$

$$dark_l_1$$

$$dark_l_2$$

```

light_l1
light_l2
live_outside
live_l1 ← live_w0
live_w0 ← live_w1 ∧ up_s2 ∧ ok_s2
live_w0 ← live_w1 ∧ down_s2 ∧ ok_s2
live_w1 ← live_w2 ∧ up_s1 ∧ ok_s1
live_w2 ← live_w3 ∧ down_s1 ∧ ok_s1
live_l2 ← live_w4
live_w4 ← live_w3 ∧ up_s3 ∧ ok_s3
live_p1 ← live_w3
live_w3 ← live_w5 ∧ ok_cb1
live_p2 ← live_w4
live_w5 ← live_w6 ∧ ok_cb2
live_w6 ← live_outside
lit_l1 ← light_l1 ∧ live_l1 ∧ ok_l1
lit_l2 ← light_l2 ∧ live_l2 ∧ ok_l2
false ← dark_l1 ∧ lit_l1
false ← dark_l2 ∧ lit_l2
assumable ok_cb1, ok_cb2, ok_s1, ok_s2, ok_s3, ok_l1, ok_l2

```

图 5-8 例 5-20 的知识

188

给定图 5-8 所示的知识和观察值，有两个最小冲突存在，即

$\{ok_cb1, ok_s1, ok_s2, ok_l1\}$ 和 $\{ok_cb1, ok_s3, ok_l2\}$

因此有

$KB \models \neg ok_cb1 \vee \neg ok_s1 \vee \neg ok_s2 \vee \neg ok_l1$

$KB \models \neg ok_cb1 \vee \neg ok_s3 \vee \neg ok_l2$

这意味着 $cb1$ 、 $s1$ 、 $s2$ 或者 $l1$ 中至少有一个组件必须是不正常的，而且 $cb1$ 、 $s3$ 或者 $l2$ 中至少有一个组件是不正常的。

给定了所有冲突的集合，用户可以通过系统诊断判断可能的错误。然而，对于给定的冲突集合，如果所有的冲突都可以通过几个缺陷解释，那么常常很难判定到底哪里出现了错误。通常用户想要知道是否所有冲突都可以通过一个或者两个缺陷来解释。

给定一个冲突集合，**基于一致性的诊断**是一个假说的集合，该集合至少有每个冲突中的一个元素。**最小诊断**(minimal diagnosis)是没有其子集也为诊断的诊断。对于其中一个诊断，在建模的过程中它的所有元素必须为假。

【例 5-21】 让我们继续考虑例 5-20。由于两个冲突的否定的析取是子句的逻辑结果，则以下合取

$(\neg ok_cb1 \vee \neg ok_s1 \vee \neg ok_s2 \vee \neg ok_l1) \wedge (\neg ok_cb1 \vee \neg ok_s3 \vee \neg ok_l2)$

是遵循知识库的。这种析取的合取可以分配成**析取范式**(Disjunctive Normal Form, DNF)，否定原子的合取的析取为：

$\neg ok_cb1 \vee$
 $(\neg ok_s1 \wedge \neg ok_s3) \vee (\neg ok_s1 \wedge \neg ok_l2) \vee$
 $(\neg ok_s2 \wedge \neg ok_s3) \vee (\neg ok_s2 \wedge \neg ok_l2) \vee$
 $(\neg ok_l1 \wedge \neg ok_s3) \vee (\neg ok_l1 \wedge \neg ok_l2)$

因此，或者是 $cb1$ 是坏的或者 6 个双重错误中至少存在一个。

分离的命题共同对应了 7 个最小诊断： $\{ok_cb1\}$ ， $\{ok_s1, ok_s3\}$ ， $\{ok_s1, ok_l2\}$ ，

$\{ok_{s_2}, ok_{s_3}\}, \{ok_{s_2}, ok_{l_2}\}, \{ok_{l_1}, ok_{s_3}\}, \{ok_{l_1}, ok_{l_2}\}$ 。系统已经被证明这些组合中至少有一个一定有错误。

189

5.4.4 通过假设和 Horn 子句推理

为了在 Horn 子句知识库中寻找冲突，本节将介绍自底向上和的自顶向下实现方法。

1. 自底向上的实现

自底向上的实现是在 5.2.2 节中确定子句的自底向上算法的增强版。

对于该算法的修改为：结论是成对的 $\langle a, A \rangle$ ，其中 a 是原子， A 是一个假说集合，该集合说明了 a 在 Horn 子句知识库 KB 的上下文环境中。

最初，结论集合 C 是 $\{\langle a, \{a\} \rangle : a \text{ 是假说}\}$ 。子句可以被用来推导新的结论。如果有一个子句 $h \leftarrow b_1 \wedge \dots \wedge b_m$ ，那么对于每一个 b_i ，都存在一些 A_i ，使得 $\langle b_i, A_i \rangle \in C$ ，则 $\langle h, A_1 \cup \dots \cup A_m \rangle$ 可以被加入到 C 中。注意，这包含了原子子句的情况，当 $m=0$ 时， $\langle h, \{\} \rangle$ 被加入到 C 中。图 5-9 对于这个算法给出了伪代码。

当 $\langle false, A \rangle$ 生成时，假设 A 形成一个冲突。

```

1: procedure ConflictBU(KB, Assumables)
2:   Inputs
3:     KB: Horn 子句集合
4:     Assumables: 可以被假设的原子集合
5:   Output
6:     冲突集合
7:   Local
8:      $C$  是一个原子和假说集合成对出现的集合
9:      $C := \{\langle a, \{a\} \rangle : a \text{ 是假说}\}$ 
10:  repeat
11:    select  $KB$  中的子句 " $h \leftarrow b_1 \wedge \dots \wedge b_m$ ", 满足
12:      对所有  $i, \langle b_i, A_i \rangle \in C$ , 且
13:       $\langle h, A \rangle \notin C$ , 其中  $A = A_1 \cup \dots \cup A_m$ 
14:       $C := C \cup \{\langle h, A \rangle\}$ 
15:  until 没有更多可能的选择
16:  return  $\{A : \langle false, A \rangle \in C\}$ 

```

图 5-9 计算冲突自底向上验证程序

我们使用修剪假设超集的方法对该程序进行细化。如果 $\langle a, A_1 \rangle$ 和 $\langle a, A_2 \rangle$ 在 C 中，且 $A_1 \subset A_2$ ，那么 $\langle a, A_2 \rangle$ 可以从 C 中移除或不加入到 C 中。没有任何理由需要使用额外的假说来说明 a 。同样，如果 $\langle false, A_1 \rangle$ 和 $\langle a, A_2 \rangle$ 在 C 中，且 $A_1 \subseteq A_2$ ，那么 $\langle a, A_2 \rangle$ 可以从 C 中移除，因为 A_1 和任何超集（包括 A_2 ）都与给定子句矛盾，因此从这样的假说集合中学习不到任何知识。

190

【例 5-22】考虑图 5-8 的公理化形式，讨论例 5-20。

首先，在图 5-9 算法中， C 有以下值：

$\{\langle ok_{l_1}, \{ok_{l_1}\} \rangle, \langle ok_{l_2}, \{ok_{l_2}\} \rangle, \langle ok_{s_1}, \{ok_{s_1}\} \rangle, \langle ok_{s_2}, \{ok_{s_2}\} \rangle, \langle ok_{s_3}, \{ok_{s_3}\} \rangle, \langle ok_{cb_1}, \{ok_{cb_1}\} \rangle, \langle ok_{cb_2}, \{ok_{cb_2}\} \rangle\}$

下面展示了在一个选择序列前提下加入到 C 的序列值：

$\langle live_outside, \{\} \rangle$
 $\langle connected_to_w_5, outside, \{\} \rangle$
 $\langle live_w_5, \{\} \rangle$
 $\langle connected_to_w_3, w_5, \{ok_{cb_1}\} \rangle$
 $\langle live_w_3, \{ok_{cb_1}\} \rangle$
 $\langle up_s_3, \{\} \rangle$
 $\langle connected_to_w_4, w_3, \{ok_{s_3}\} \rangle$
 $\langle live_w_4, \{ok_{cb_1}, ok_{s_3}\} \rangle$
 $\langle connected_to_l_2, w_4, \{\} \rangle$
 $\langle live_l_2, \{ok_{cb_1}, ok_{s_3}\} \rangle$
 $\langle light_l_2, \{\} \rangle$

$\langle lit_l_2, \{ok_cb_1, ok_s_3, ok_l_2\} \rangle$
 $\langle dark_l_2, \{\} \rangle$
 $\langle false, \{ok_cb_1, ok_s_3, ok_l_2\} \rangle$

因此, 知识库蕴含:

$\neg ok_cb_1 \vee \neg ok_s_3 \vee \neg ok_l_2$

其他冲突可以通过后续的算法找到。

2. 自顶向下的实现

自顶向下的实现方法与图 5-4 注释描述的自顶向下确定子句类似, 除了顶层目标是为了证明 *false*, 而且验证中所遇到的假说不会被证明, 但是会被收集起来。

191

图 5-10 给出了寻找冲突的算法。不同的选择可能会导致找到不同的冲突。如果没有选择可用, 则算法失败。

【例 5-23】 考虑例 5-20 中所示的电路。对于一个导致冲突的选择和决策序列, 以下是 *G* 的值的序列:

$\{false\}$
 $\{dark_l_1, lit_l_1\}$
 $\{lit_l_1\}$
 $\{light_l_1, live_l_1, ok_l_1\}$
 $\{live_l_1, ok_l_1\}$
 $\{live_w_0, ok_l_1\}$
 $\{live_w_1, up_s_2, ok_s_2, ok_l_1\}$
 $\{live_w_3, up_s_1, ok_s_1, up_s_2, ok_s_2, ok_l_1\}$
 $\{live_w_5, ok_cb_1, up_s_1, ok_s_1, up_s_2, ok_s_2, ok_l_1\}$
 $\{live_outside, ok_cb_1, up_s_1, ok_s_1, up_s_2, ok_s_2, ok_l_1\}$
 $\{ok_cb_1, up_s_1, ok_s_1, up_s_2, ok_s_2, ok_l_1\}$
 $\{ok_cb_1, ok_s_1, up_s_2, ok_s_2, ok_l_1\}$
 $\{ok_cb_1, ok_s_1, ok_s_2, ok_l_1\}$

集合 $\{ok_cb_1, ok_s_1, ok_s_2, ok_l_1\}$ 作为一个冲突返回。选择使用不同的子句将会导致不同的结果。

```

1: non-deterministic procedure conflictTD(KB, Assumables)
2:   Inputs
3:     KB: Horn 子句集合
4:     Assumables: 可以被假定的原子集合
5:   Output
6:     一个冲突
7:   Local
8:     G 是一个原子集合 (暗示为假)
9:     G := { false }
10:  repeat
11:    select G 中的  $a, a \notin Assumables$ 
12:    choose KB 中以  $a$  为头部的子句 C
13:    以在 C 的主体中的原子替代 G 中的  $a$ 
14:  until  $G \subseteq Assumables$ 
15:  return G
  
```

图 5-10 自顶向下寻找冲突的 Horn 子句解释程序

5.5 完备知识假设

当一个数据库中任何没有被阐明的都为假, 从意义上讲这个数据库往往是完备的。

【例 5-24】 你可能希望用户指出哪些开关是置上的, 哪些断路器是坏掉的, 通过上述系统可以推断出那些没有被说明为置上的开关是置下的, 没有被说明为坏掉的断路器是正常的。因此, 置下是开关的默认设置, 状态良好是断路器的默认设置。对于用户来说, 相对于指定哪些开关是置下的和哪些断路器状态良好等看似多余的信息来说, 使用默认设置可以很方便地进行交互。为了使用这样的默认设置进行推理, 必须假设一个 Agent 拥有完全知识, 开关的状态没有被指出因为它的状态是置下的, 而不是因为 Agent 不知道它是置下还是置上的。

在之前给出的确定子句逻辑中, 不允许从缺失的知识或者失败的验证中得出结论的推导。该逻辑没有假设知识是完备的。特别是一个原子的否定不可能作为确定子句知识库的

192

一个逻辑结果。

完备知识假设(complete knowledge assumption)是假设对于任何原子, 当该原子为真时, 拥有该原子作为头部的子句覆盖了所有的情况。基于这种假设, 如果 Agent 不能确定一个原子为真则该原子为假。这也被称为**封闭世界假设**(closed-world assumption)。这可以对比于**开放世界假设**(open-world assumption), 开放世界假设使 Agent 一无所知, 所以无法从缺失的数据库中得到任何结论。封闭世界假设需要 Agent 知道该世界的所有相关知识。

一个原子存在规则时, 假设则是每个原子的规则覆盖了原子为真的所有情况。特别是, 假设原子 a 的规则是:

$$a \leftarrow b_1$$

...

$$a \leftarrow b_n$$

其中原子子句 a 表示规则 $a \leftarrow true$ 。完备知识假设意味着如果在某些解释中 a 为真, 那么必有一个 b_i 在该解释中一定为真; 也就是说,

$$a \rightarrow b_1 \vee \dots \vee b_n$$

由于定义 a 的子句等价于

$$a \leftarrow b_1 \vee \dots \vee b_n$$

该子句的意义可以视为联合上述两个命题, 即等价于

$$a \leftrightarrow b_1 \vee \dots \vee b_n$$

其中 \leftrightarrow 读作“当且仅当”(见图 5-1)。对于 a 这种等效被称为子句的 **Clark 完备化**(Clark's completion)。一个知识库的 Clark 完备化是对于在该知识库中每一个原子的完备。

Clark 完备化意味着如果对于一个原子 a 没有规则, 那么该原子的完备是 $a \leftrightarrow false$, 这意味着 a 为假。

【例 5-25】 考虑例 5-5 中的如下子句:

$$down_{s_1}$$

$$up_{s_2}$$

$$live_{l_1} \leftarrow live_{w_0}$$

$$live_{w_0} \leftarrow live_{w_1} \wedge up_{s_2}$$

$$live_{w_0} \leftarrow live_{w_2} \wedge down_{s_2}$$

$$live_{w_1} \leftarrow live_{w_3} \wedge up_{s_1}$$

假设仅有以原子作为头部的子句, 且没有子句 up_{s_1} 和 $down_{s_2}$ 。这些原子的完备集是

$$down_{s_1} \leftrightarrow true$$

$$up_{s_1} \leftrightarrow false$$

$$up_{s_2} \leftrightarrow true$$

$$down_{s_2} \leftrightarrow false$$

$$live_{l_1} \leftrightarrow live_{w_0}$$

$$live_{w_0} \leftrightarrow (live_{w_1} \wedge up_{s_2}) \vee (live_{w_2} \wedge down_{s_2})$$

$$live_{w_1} \leftrightarrow live_{w_3} \wedge up_{s_1}$$

这暗示了 up_{s_1} 为假, 且 $live_{w_1}$ 为假。

通过完备性, 系统可以得出否定的结论, 且扩展语言允许子句主体中的存在否定是很有用的。**文字**(literal)既可以是一个原子也可以是原子的否定。一个确定子句的定义可以被扩展, 该扩展允许主体中存在文字而不仅仅是原子。在完全知识假设中, 记原子 a 的否定为 $\sim a$, 以此来区别它与并不假设完备性的传统否定。这种否定经常被称为**否定即失败**

(negation as failure)。

基于否定即失败, 如果 $KB' \models g$, 则主体 g 是知识库 KB 的结果, 其中 KB' 是 KB 的 Clark 完备化。在子句主体或者查询中的否定 $\sim a$ 在完备集中变为 $\neg a$ 。也就是说, 若查询为基于完备知识假设下的知识库中的查询, 则意味着该查询是该知识库的完备集的逻辑结果。

【例 5-26】考虑例 5-5 的公理化。通过用户告诉系统哪些开关是置上的, 而如果没有被告知哪些开关是置上的, 则可通过系统总结哪些开关是置下的。这种方法可以更简单地表示该领域的知识。这可以通过增加以下规则实现:

$down_s_1 \leftarrow \sim up_s_1$

$down_s_2 \leftarrow \sim up_s_2$

$down_s_3 \leftarrow \sim up_s_3$

同样, 除非其被告知断路器是坏掉的, 否则系统可以总结出断路器是良好的。

$ok_cb_1 \leftarrow \sim broken_cb_1$

$ok_cb_2 \leftarrow \sim broken_cb_2$

虽然这看起来比之前的表示更加复杂, 但是这意味着在一个特定情形下用户可以更容易地指明当前情况。用户仅仅需要指定哪些是置上的, 哪些是坏的。如果置下是开关的常态, 良好是断路器的常态, 这会节省很多时间。

为了表示图 5-2 的状态, 用户指定

up_s_2

up_s_3

系统可以推断, s_1 一定是置下的, 而且两个断路器状态都为良好。

完备知识库为

$down_s_1 \leftrightarrow \neg up_s_1$

$down_s_2 \leftrightarrow \neg up_s_2$

$down_s_3 \leftrightarrow \neg up_s_3$

$ok_cb_1 \leftrightarrow \neg broken_cb_1$

$ok_cb_2 \leftrightarrow \neg broken_cb_2$

$up_s_1 \leftrightarrow false$

$up_s_2 \leftrightarrow true$

$up_s_3 \leftrightarrow true$

$broken_cb_1 \leftrightarrow false$

$broken_cb_2 \leftrightarrow false$

注意, 在子句主体中但不是任何子句头部的原子在完备集中都为假。

回想一下, 若对原子都分配一个自然数, 且子句主体中的原子比头部的原子所分配的数更小, 则称知识库是无环(acyclic)的。通过否定即失败, 非无环知识库在语义上就会出现问

下面的知识库不是无环的:

$a \leftarrow \sim b$

$b \leftarrow \sim a$

这个知识库的数据库完备集等价于 $a \leftrightarrow \neg b$, 其指定 a 和 b 所取的真值不同, 但并未指定哪一个为真。

下面的知识库也不是无环的:

$a \leftarrow \sim a$

该知识库的 Clark 完备集是 $a \leftrightarrow \neg a$, 其是逻辑矛盾的。

对于一个无环知识库的 Clark 完备化常常是一致的, 且常给出每个原子的一个真值。

对于本章剩下的部分，我们假设知识库是无环的。

5.5.1 非单调推理

在某子句加入前所能总结的任何结论在加入该子句后仍然可以被总结出来，加入知识并不减少可以被导出的命题集合，则称该确定子句逻辑是单调的(monotonic)。

如果增加了更多的知识后，某些结论就失效了，则该逻辑是非单调的(non-monotonic)。否定即失败的确定子句的逻辑是非单调的。非单调推理对于表示默认设置是很有效的。默认设置(default)是指除被某异常重写外，都可使用的规则。

比如，如果 c 是真， b 在一般情况下为真，则知识库设计者可以编写如下形式的规则：

$$b \leftarrow c \wedge \sim ab_a$$

其中， ab_a 是一个原子，表示面对 a 的某些表达时存在异常。给定 c ，Agent 可以推断 b ，除非它被告知 ab_a 。增加 ab_a 到知识库中可以阻碍结论 b 。含有 ab_a 的规则可以用来阻碍规则主体条件的默认设置。

【例 5-27】 假设采购 Agent 调查度假情况。度假酒店可以是毗邻海滩或者远离海滩。如果该度假酒店毗邻沙滩，知识提供者可能会特别指出，因此可以获得以下子句：

$$away_from_beach \leftarrow \sim on_beach$$

如果该 Agent 没有被告知酒店靠近海滩，则 Agent 可以通过该子句推断出酒店远离海滩。

协作系统(cooperative system)试图不产生误导。如果被告知酒店在海滩上，则知道酒店用户可到达海滩。如果他们已到达海滩，则期望用户可以在海滩上游泳。基于此有如下默认设置：

$$beach_access \leftarrow on_beach \wedge \sim ab_{beach_access}$$

$$swim_at_beach \leftarrow beach_access \wedge \sim ab_{swim_at_beach}$$

如果沙滩上的酒店没有沙滩通道或者不可以游泳，那么协作系统将会告知。我们也会指出，如果存在一个封闭海湾或者大城市，则就不可以游泳，通过默认设置定义如下：

$$ab_{swim_at_beach} \leftarrow enclosed_bay \wedge big_city \wedge \sim ab_{no_swimming_near_city}$$

可以说不列颠哥伦比亚对于靠近城市游泳是异常的，则

$$ab_{no_swimming_near_city} \leftarrow in_BC \wedge \sim ab_{BC_beaches}$$

仅鉴于前面给定规则，Agent 推断 $away_from_beach$ 。如果接下来被告之其 on_beach ，亦可以推断其不会 $away_from_beach$ ，但是可以推断其 $beach_access$ 并且 $swim_at_beach$ 。如果其也被告知了 $enclose_bay$ 和 big_city ，则其就不会被推断 $swim_at_beach$ 。然而，如果其接下来被告知 in_BC ，那么可以推断 $swim_at_beach$ 。

通过对正常情况下赋予默认设置，用户可以通过告诉其什么是异常的来与系统进行交互，这在通信中非常经济。用户不必对明显的事物进行陈述。

非单调性推理的一种思考方式是进行变元(argument)。规则可以被用来作为论证的组件，对否定异常给出了一种削弱论证的方式。注意，在提出的语言中，只有已存在的正面论证可以被削弱。在更泛化的理论中，正面论证和负面论证可以互相攻击。

5.5.2 完备知识的验证程序

1. 自底向上程序

否定即失败的自底向上程序是对确定子句自底向上程序的修改。区别是它可以在被导

出的结果集合 C 中加入形如 $\sim p$ 的文字, 当可以确定 p 一定失败时, 可以将 $\sim p$ 加入 C 中。

失败可以递归地进行定义: 当以 p 为头部的子句的每个主体都失败时则 p 是失败的。如果主体中有一个文字是失败的, 则该主体是失败的。如果已经导出 $\sim b_i$, 则主体中的原子 b_i 是失败的。如果已经导出 b_i , 则主体中的否定 $\sim b_i$ 是失败的。

图 5-11 对于 KB 的计算结果, 给出了一个自底向上否定即失败的解释程序。注意这包含了子句的主体为空的情况(其中 $m=0$, 并且在头部的原子被加入了 C 中), 而且存在某一原子并没有出现在任何一个子句的头部的情况(这种情况中, 其否定被加入 C 中)。

【例 5-28】 考虑如下子句:

$p \leftarrow q \wedge \sim r$

$p \leftarrow s$

$q \leftarrow \sim s$

$r \leftarrow \sim t$

t

$s \leftarrow w$

其次是一组加入 C 中的可能文字序列

t ,

$\sim r$,

$\sim w$,

$\sim s$,

q ,

p ,

其中, 由于 t 是作为原子子句给定的, 所以 t 是平凡导出的; 由于 $t \in C$, 导出 $\sim r$; 由于没有子句 w , 导出 $\sim w$, 所以在图 5-11 第 18 行“每一个子句”的条件平凡持有。由于 $\sim w \in C$, 导出文字 $\sim s$; 由于主体全部被证明了, 导出 p 和 q 。

2. 自顶向下的否定即失败程序

对于完备知识假设的自顶向下程序通过否定即失败产生。这与图 5-4 自顶向下确定子句验证程序相似。这是一个非确定性程序, 其可以通过搜索成功的选择来实现。当选择了一个否定原子 $\sim a$, 则对于原子 a 的一个新的验证就开始了。如果对于 a 验证失败了, 则 $\sim a$ 成功。如果对于 a 的验证成功了, 则算法失败, 必须进行其他的选择。算法如图 5-12 所示。

【例 5-29】 考虑例 5-28 的子句, 假设查询是 $\text{ask } p$ 。

初始化 $G = \{p\}$ 。

使用对于 p 的第一条规则, G 变为 $\{q, \sim r\}$ 。

选择 q , 并将它与第三条规则的主体替换, G 变为 $\{\sim s, \sim r\}$ 。

接下来选择 $\sim s$, 并开始验证 s 。验证 s 失败, 因此 G 变为 $\{\sim r\}$ 。

接下来选择 $\sim r$, 并开始验证 r 。对于验证 r , 有子目标 $\sim t$, 因此需要尝试验证 t 。对于 t 的验证成功。因此对于 $\sim t$ 的验证失败, 由于对于 r 没有更多的规则, 则对 r 的验证失败。因此, 对于 $\sim r$ 的验证成功。

```

1: procedure NAFBU(KB)
2:   Inputs
3:      $KB$ : 可以包含否定即失败的子句集合
4:   Output
5:     产生于  $KB$  的完备集的文字集合
6:   Local
7:      $C$  为文字集合
8:      $C := \{\}$ 
9:   repeat
10:    either
11:      select  $r \in KB$ , 满足
12:         $r$  是 " $h \leftarrow b_1 \wedge \dots \wedge b_m$ "
13:        对于所有  $i, b_i \in C$ , 且
14:         $h \notin C$ ;
15:         $C := C \cup \{h\}$ 
16:    or
17:      select  $h$ , 满足  $\sim h \notin C$  且
18:        其中每一个子句 " $h \leftarrow b_1 \wedge \dots \wedge b_m$ "  $\in KB$ 
19:        对于某个  $b_i, \sim b_i \in C$ 
20:        或者某个  $b_i = \sim g$  且  $g \in C$ 
21:         $C := C \cup \{\sim h\}$ 
22:  until 没有更多的可选原子
  
```

图 5-11 自底向上否定即失败验证程序

G 为空, 所以为顶层查询返回 *yes*。

注意这实现了有限失败 (finite failure), 因为如果验证程序没有停止, 则其就没有结论。比如, 假设存在一个规则 $p \leftarrow p$ 。ask p 的查询就不能停止。完整性 $p \leftrightarrow p$ 没有给出任何信息。即使可能存在一种方法得出永远不会有 p 的验证, 但由于其并没有遵循完备性, 合理的验证程序也不能得出 $\sim p$ 的结论。

```

1: non-deterministic procedure NAFTD( $KB, Query$ )
2:   Inputs
3:      $KB$ : 可以包含否定即失败的子句集合
4:      $Query$ : 需要验证的文字集合
5:   Output
6:     如果  $KB$  的完整性蕴含  $Query$  返回 yes, 否则返回 no
7:   Local
8:      $G$  是一个文字集合
9:      $G := Query$ 
10:  repeat
11:    select 文字  $l \in G$ 
12:    if  $l$  是  $\sim a$  的形式 then
13:      if NAFTD( $KB, a$ ) 失败 then
14:         $G := G \setminus \{l\}$ 
15:      else
16:        失败
17:    else
18:      choose  $KB$  中子句  $l \leftarrow B$ 
19:      在  $G$  中用  $B$  替代  $l$ 
20:  until  $G = \{\}$ 
21:  return yes

```

图 5-12 自顶向下否定即失败解释程序

5.6 溯因推理

溯因推理 (abduction) 是一种推理的形式, 其中假设是用来解释观察的。比如, 如果 Agent 观察到某些灯没有工作, 则其可以推测在该世界中发生了什么来解释为什么灯没有工作。在智能教学系统中, 可以尝试通过学生所理解的知识及其不理解的知识, 解释为什么学生给出了那样一些答案。

溯因推理一词是由 Peirce (1839—1914) 提出的, 用于使这种推理类型区别于演绎和归纳。演绎 (deduction) 是确定哪些命题是逻辑跟随于一组公理, 而归纳 (induction) 则是从例子中推断一般性关系。

在溯因推理中, Agent 基于其观察到的情况推测哪些内容可能是正确的。Agent 确定其观察所蕴含的内容——当何种可能为真时会使得观察为真。观察是隐含于矛盾之中的 (因此矛盾逻辑蕴含所有), 所以我们希望从我们的观察的解释中找到矛盾。

为了形式化溯因推理, 我们使用 Horn 子句和假说 (与反证法证明使用相同的输入)。给出了系统:

- 知识库 KB , 是一个 Horn 子句的集合。
- 假说 (assumable) 原子集合 A , 假说是假设的基础。

不将观察加入知识库, 而是需要解释观察。

$\langle KB, A \rangle$ 的情景 (scenario) 是 A 的子集 H , 使得 $KB \cup H$ 是可满足的。如果一个模型

存在, 且其中每一个 KB 的元素和 H 的每一个元素都是真的, $KB \cup H$ 就是可满足的。如果 H 的所有子集都不是 KB 的冲突, 则 $KB \cup H$ 就是可满足的。

对于在 $\langle KB, A \rangle$ 中的命题 g 的解释(explanation)是一个情景, 该情景与 KB 一起实现 g 。也就是说, 对于命题 g 的解释是一个集合 H , $H \subseteq A$ 使得

$KB \cup H \models g$

$KB \cup H \not\models \text{false}$

对于 $\langle KB, A \rangle$ 中, 如果 H 是 $\langle KB, A \rangle$ 中 g 的解释, 且没有一个严格 H 的严格子集仍然是 $\langle KB, A \rangle$ 中的 g 的解释, 那么 H 是 g 的最小解释(minimal explanation)。

【例 5-30】 作为一个诊断助手, 考虑下面的简单知识库和假设:

支气管炎 \leftarrow 流感

支气管炎 \leftarrow 吸烟

咳嗽 \leftarrow 支气管炎

气喘 \leftarrow 支气管炎

发烧 \leftarrow 流感

喉咙痛 \leftarrow 流感

假 \leftarrow 吸烟 \wedge 非吸烟者

假设: 吸烟, 非吸烟者, 流感

如果 Agent 观察到气喘, 有两种最小解释:

{流感} 和 {吸烟}

这些解释隐含了支气管炎和咳嗽。

如果观察到了气喘 \wedge 发烧, 则只有一个最小解释:

{流感}

另一个解释在最小解释中就不再需要了。

注意, 当观察到气喘时, Agent 推理一定是支气管炎, 并且流感和吸烟是假设的罪魁祸首。然而, 如果也观察到了发烧, 则病人一定是得了流感, 所以就没有必要假设其吸烟, 它已经被解释消除(explained away)了。

如果观察到的是气喘 \wedge 非吸烟者, 那只有一种最小解释:

{流感, 非吸烟者}

另一个气喘的解释与非吸烟者这个条件不一致。

基于行为的观察来确定在系统内部将会发生的事情是诊断(diagnosis)或确认(recognition)问题。在溯因诊断(abductive diagnosis)中, Agent 猜测疾病、障碍和正常工作的部件来解释所观察到的症状。这与基于一致性的诊断不同, 在一致性诊断中设计者不仅对正常行为建模也对错误行为建模, 观察被用来解释而不是加入知识库中。由于知识库必须可以实际地证明观察, 溯因推理诊断需要更加细致地建模并且给出更加详细的诊断。它也允许 Agent 来诊断没有正常表现的系统。比如, 一个智能教学系统中, 通过观察学生的行为, 该教学系统可以推测出哪些知识学生理解了, 哪些没有理解, 这可以引导教学系统的行为。

溯因推理也可以用来进行设计(design), 需要进行解释的事物是该设计的目标, 而假设则是该设计的基石。这种设计就是一种解释。一致性意味着设计是可行的。设计目标意味着该设计达到了预期设计目标。

【例 5-31】 考虑图 5-2 中的电气领域系统。与在例 5-20 中提出的一致性诊断例子的表示一样, 公理化可能在系统中发生的假设。在溯因诊断中, 必须公理化既从缺陷又从一般

性推理两个方面得出的内容。对于每一个可以被观察的原子，都需要公理化其是如何产生的。在接下来的例子中，假说原子使用“假说”关键字内联声明。

用户可以观察 l_1 是明或者暗，我们必须表示出公理化系统中使该观察为真的规则。如果它状态良好且有电力供应，则灯 l_1 是明的。如果灯是坏的或者没有电，则其为暗的。系统可以假设 l_1 是好的或者坏的，但是不能两个同时假设：

$lit_l_1 \leftarrow live_w_0 \wedge ok_l_1$

$dark_l_1 \leftarrow broken_l_1$

$dark_l_1 \leftarrow dead_w_0$

assumable ok_l_1

assumable $broken_l_1$

$false \leftarrow ok_l_1 \wedge broken_l_1$

电线 w_0 是通路或者断路基于开关的位置和进入的电线是通或者断：

$live_w_0 \leftarrow live_w_1 \wedge up_s_2 \wedge ok_s_2$

$live_w_0 \leftarrow live_w_2 \wedge down_s_2 \wedge ok_s_2$

$dead_w_0 \leftarrow broken_s_2$

$dead_w_0 \leftarrow up_s_2 \wedge dead_w_1$

$dead_w_0 \leftarrow down_s_2 \wedge dead_w_2$

assumable ok_s_2

assumable $broken_s_2$

$false \leftarrow ok_s_2 \wedge broken_s_2$

同样可以公理化其他的电线。一些电线依赖于断路器是正常的或是损坏的：

$live_w_3 \leftarrow live_w_5 \wedge ok_cb_1$

$dead_w_3 \leftarrow broken_cb_1$

$dead_w_3 \leftarrow dead_w_5$

assumable ok_cb_1

assumable $broken_cb_1$

$false \leftarrow ok_cb_1 \wedge broken_cb_1$

对于这个问题的剩余部分，我们假设其他的灯和电线也类似的表示。

外部的电源可以是通的，也可以是断的：

$live_w_5 \leftarrow live_outside$

$dead_w_5 \leftarrow outside_power_down$

assumable $live_outside$

assumable $outside_power_down$

$false \leftarrow live_outside \wedge outside_power_down$

开关可以假设为置上或置下：

assumable up_s_1

assumable $down_s_1$

$false \leftarrow up_s_1 \wedge down_s_1$

对于 lit_l_1 有两种最小解释：

$\{live_outside, ok_cb_1, ok_l_1, ok_s_1, ok_s_2, up_s_1, up_s_2\}$

$\{down_s_1, down_s_2, live_outside, ok_cb_1, ok_l_1, ok_s_1, ok_s_2\}$

在设计阶段可以看出，作为一种保证灯是亮的方法：把两个开关都置上或者两个都置下，并保证所有的开关都正常工作。如果 Agent 观察到 l_1 为亮的，可用该方法确定发生的事情。上述两种情况都必须满足。

对于 $dark_l_1$ 有 10 种最小解释:

```
{broken_l1}
{broken_s2}
{down_s1, up_s2}
{broken_s1, up_s2}
{broken_cb1, up_s1, up_s2}
{outside_power_down, up_s1, up_s2}
{down_s2, up_s1}
{broken_s1, down_s2}
{broken_cb1, down_s1, down_s2}
{down_s1, down_s2, outside_power_down}
```

203

对于 $dark_l_1 \wedge lit_l_2$ 有 6 种最小解释:

```
{broken_l1, live_outside, ok_cb1, ok_l2, ok_s3, up_s3}
{broken_s2, live_outside, ok_cb1, ok_l2, ok_s3, up_s3}
{down_s1, live_outside, ok_cb1, ok_l2, ok_s3, up_s2, up_s3}
{broken_s1, live_outside, ok_cb1, ok_l2, ok_s3, up_s2, up_s3}
{down_s2, live_outside, ok_cb1, ok_l2, ok_s3, up_s1, up_s3}
{broken_s1, down_s2, live_outside, ok_cb1, ok_l2, ok_s3, up_s3}
```

注意解释不能包括 $outside_power_down$ 或者 $broken_cb1$, 因为它们对于解释 l_2 是明状态来说是矛盾的。

溯因推理也可使用基于假设推理的 Horn 子句中自底向上和自顶向下的实现方法。图 5-9 的自底向上的实现方法计算 C 中每一个子句的最小解释。也可以使用在之前已经讨论的修剪方法。自顶向下实现可以通过使用相同的编码和知识库产生矛盾来找到所有 g 的解释, 证明 g 而不证明 $false$ 。 g 的最小解释是最小假说集合, 该假说集合被收集用于证明 g 为非冲突子集。

5.7 因果模型

原始(primitive)原子是当其为真时, 标记为原子子句的原子。**导出(derived)**原子是当其为真时, 使用规则来定义的原子。特别地, 设计者为导出原子定义公理并且希望用户来指出哪些原始原子是真的。因此, 导出原子会从原始原子和其他可以被导出的原子中推导出来。

Agent 的设计者必须在设计一个领域知识库时做出很多决定。比如, 考虑两个都为真的命题, a 和 b 。对于如何定义它们有很多种选择。设计者可以指定 a 和 b 都是原子子句, 并都作为原始原子。设计者也可指定 a 为原始原子, b 为导出原子, 标记 a 为原子子句并给定规则 $b \leftarrow a$ 。另外, 设计者可以指定原子子句 b 和规则 $a \leftarrow b$, 将 b 作为原始原子, a 作为导出原子。这些表达都是逻辑等价的, 它们不能从逻辑上被区分。然而, 在当知识库改变的时候它们会产生不同的效果。假设 a 对于一些推理来说不再是为真, 在第一和第三种表示中, b 将仍然会是真的, 但在第二种表示中 b 就不再是真的了。

因果模型(causal model)或者是具有**因果关系(causality)**的模型, 是某一知识域的代表, 其可以预测干预的效果。**干预(intervention)**是指对某一变量强制赋予一个特殊的值, 也就是说, 在模型中干预可以不利用其他变量而改变值。

204

为了预测干预的效果, 因果模型表示了原因如何蕴含它的效果。当原因被改变时, 干

预的效果会改变。**证据模型**(evidential model)展示了领域知识的另一个方向——从效果到原因。注意我们不是假设一个效果存在原因,而是有很多联合的命题使得效果为真。

【例 5-32】 考虑图 1-8 描绘的电气系统。在这个系统中,开关 s_3 是置上的且灯 l_2 是亮的。有很多种方法可以公理化这个系统。例 5-5 包括了如下因果规则:

$$lit_l_2 \leftarrow up_s_3 \wedge live_w_3$$

此外,我们可以指定证据方向:

$$up_s_3 \leftarrow lit_l_2$$

$$live_w_3 \leftarrow lit_l_2$$

系统中的所有陈述都为真。

假设电线 w_3 是通路且有人将开关 s_3 置上了,我们可以导出 l_2 将会点亮。然而,如果有人通过一些模型外部的设备(而不是通过翻转开关)将 s_3 点亮,我们就不会推理出开关作为副作用而被置上。

【例 5-33】 考虑图 1-8 描述的电气系统。下面的命题描述了关于开关 s_1 、 s_2 和灯 l_1 之间的不变关系,假设所有组件都工作正常:

$$up_s_1 \leftarrow (lit_l_1 \leftrightarrow up_s_2) \quad (5-1)$$

这个公式是指三个命题之间的对称关系,当且仅当有奇数个命题为真时,公式为真。然而在现实世界中,这些命题之间的关系是不对称的。假设在某一状态下所有的原子都为真。令 s_1 置下,并不能使得 s_2 置下并使 l_1 点亮。相反,将 s_1 置下会令 lit_l_1 为假,而 up_s_2 会保持不变。因此,为了预测干预的结果,我们需要比上面式 5-1 更多的命题。

因果模型为:

$$lit_l_1 \leftarrow up_s_1 \wedge up_s_2$$

$$lit_l_1 \leftarrow \sim up_s_1 \wedge \sim up_s_2$$

完成这个命题等价于式(5-1),然而,当其中一个值变化时,它可以进行合理的预测。

证据模型为:

$$up_s_1 \leftarrow lit_l_1 \wedge up_s_2$$

$$up_s_1 \leftarrow \sim lit_l_1 \wedge \sim up_s_2$$

这可以被用来解释关于是否 s_1 置上是基于 s_2 的位置和是否 l_1 是点亮的问题。其完整性等价于式(5-1)。然而,它并不准确地预测干预的结果。

因果模型(causal model)包括:

- **背景变量**(background variable)集合,常常被称为**外因变量**(exogenous variable),是由外部模型影响来确定的;
- **内因变量**(endogenous variable)集合,其作为模型的一部分来确定;
- **函数集合**,每一个内因变量对应一个函数,其指定内因变量如何由其他内因变量和外因变量来确定的。变量 X 的函数称为 X 的**因果机制**(causal mechanism)。整个函数对于每一个为背景变量的赋值必须有唯一解。

当变量是命题的时候,对于一个命题的函数可以被指定为将其作为头部的子句集合(基于完备知识假设)。知识库是无环的情况下可以得到唯一的解。

【例 5-34】 在例 5-33 中,式 5-1 可以作为 lit_l_1 的因果机制。这可以通过规则 lit_l_1 在该模型的头部指出。对于 up_s_1 和 up_s_2 会有其他的因果机制,或者在该模型中可能它们会是背景变量而不受系统控制。

干预(intervention)是通过一些机制将变量 X 强制转换为一个特定的值 v 而不改变模

型中的其他变量。可以通过将因果机制 X 替换为 $X=v$, 来观察干预的结果。为了干预强制 p 为真, 需要将源于 p 的子句替换为 p 的原子子句。为了干预强制命题 p 为假, 需要移除 p 的子句。

如果背景变量的值未知, 背景变量可以通过假说表示。观察可以通过两个阶段来实现:

- 1) 溯因推理通过背景变量来解释观察;
- 2) 通过预测来观察解释的后继。

显然, 在给定观察时溯因推理告诉我们世界是什么样的; 在给定世界是怎样的情况下, 预测告诉我们行为的结果是什么。

5.8 本章小结

- 在一个系统中, 当不确定性或模糊性不存在时, 确定子句知识库可以用来指定某个领域的原子子句和规则。
- 给定一组关于某领域的事实集合, 逻辑推理刻画了为真的事实。
- 一个合理且完备的验证程序可以用来确定知识库的逻辑推理。
- 可以使用反证法来从 Horn 子句知识库中获得推论。
- 在知识是完备的情况下(例如, 基于完备知识假设), 可以使用否定即失败方法。
- 溯因推理可用来解释观察。
- 因果模型可以预测干预的效果。

206

5.9 参考文献及进一步阅读

本书使用的命题逻辑的语义是由 Tarski[1956]提出的。对于逻辑的介绍中, 对信息的概括请参看 Copi[1982], 对更多形式化的方法请参看 Enderton[1972]和 Mendelson[1987]二人的研究内容, Bell 和 Machover[1977]等人研究了进一步的问题。对于在人工智能中使用逻辑的深入讨论请参阅《人工智能与逻辑编程手册》(Handbook of logic in Artificial Intelligence and logic Programming[Gabbay, Hogger 和 Robinson, 1993])。

Levesque[1984]描述了基于询问的知识库的方法。

Kleer, Mackworth, Reiter[1992]形式化了基于一致性的诊断。

逻辑与 Horn 子句推理的众多基础是在丰富的逻辑内容下发展起来, 这部分内容将在第 12 章进行阐述, 对其的研究是基于逻辑编程(logic programming)进行的。Robinson[1965]扩展了推理理论。Colmerauer, Kanoui, Roussel, Pasero[1973]等人和 Kowalski[1974]基于 Green[1969], Hayes[1973], Hewitt[1969]等人的前期工作, 提出了 SLD 推理。van Emden, Kowalski[1976]等人提出了不动点语义。逻辑编程中关于语义和性能的更多细节可以参看 Lloyd[1987]的研究。

对于否定即失败的工作是基于 Clark[1978]的工作展开的。Apt 和 Bol[1994]总结了使用不同技术和否定即失败的方法来处理问题。自底向上否定即失败的验证程序的基础是 Doyle[1979]的真值维护系统(truth maintenance system)。Doyle 也考虑了增加或者移除子句的情况, 见习题 5.15。而 McCarthy[1986]提倡使用默认设置推理的异常方法。

在本书中所阐述的溯因推理框架是基于 de Kleer[1986]等人提出的基于假设的真值维护系统(ATM)和[Poole, Goebel 和 Aleliunas, 1987]提出的理论发展的。溯因推理已经被用于诊断[Peng 和 Reggia, 1990]、自然语言理解[Hobbs, stickel, Appelt 和 Martin, 1993]和时序推理[Shanahan, 1989]等方面。Kakas, Kowalski 和 Toni[1993]以及 Kakas 和 Denecker[2002]等人对溯因推理进行了回顾。对于 Peirce 的工作的总结可以参看 Burch[2008]的研究内容。

Dung[1995]针对论证提出了一个抽象的框架。该框架为本领域中的许多工作提供了基础。Dung, Mancarella 和 Toni[2007]提出了一种最近论证框架。chesnevar, Maguitman 和 Loui[2000]以及 Besnard

207

和 Hunter[2008]也致力于有关论证的工作。

de Kleer[1986]使用了基于 ATMS 的自底向上 Horn 实现方法寻找解释。ATMS 是一种更加复杂的方法，其考虑了子句和假说的增加问题，而我们忽略了这个问题(见习题 5.16)。

Pearl[2000]以及 Spirtes、Glymour 和 Scheines[2000]讨论了因果模型。

5.10 习题

本章习题使用 AILog 系统，它是一种简单的逻辑推理系统，可以实现在本章中讨论的所有推理，AILog 系统可以从本书的网站获得。

5.1 假设我们想要在电气领域中，对将一个电水壶插入电源插座中的系统进行推理。假设电水壶为了进行加热，必须插入一个电源插座中，其必须打开，而且必须将水装入其中。

使用 AILog 语法，写下使得系统确定电水壶是否在加热的公理。AILog 对于电力环境的编码方式可以从本书网站获得。

你必须：

- 给出所有预计使用的符号的解释。
- 写出可以在 AILog 中加载的子句。
- 展示出 AILog 中运行得到的知识库。

5.2 考虑图 5-13 所描述的房间水管系统。

在这个例子中， p_1 、 p_2 和 p_3 表示为冷水管， t_1 、 t_2 和 t_3 表示水龙头， d_1 、 d_2 和 d_3 表示排水管，shower 表示淋浴器，bath 表示浴缸，sink 表示水槽，floor 表示地板，图 5-13 给出了符号的意义。

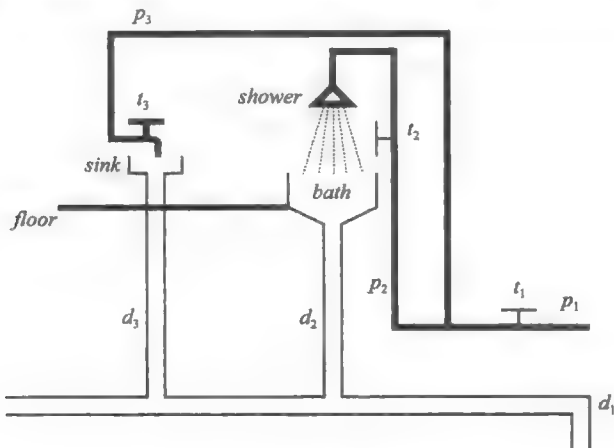


图 5-13 房间水管系统

假设我们有如下的原子：

- 如果 p_i 中有干线压力，则 $pressurized_p_i$ 为真。
- 如果水龙头 t_i 打开，则 on_t_i 为真。
- 如果水龙头 t_i 关闭，则 off_t_i 为真。
- 如果 b 是湿的，则 wet_b 为真(b 可以是水槽、浴缸或者地板)。
- 如果水流经水管 p_i ，则 $flow_p_i$ 为真。
- 如果水槽被塞住，则 $plugged_sink$ 为真。
- 如果浴缸被堵塞，则 $plugged_bath$ 为真。
- 如果水槽未被塞住，则 $unplugged_sink$ 为真。
- 如果浴缸未被堵塞，则 $unplugged_bath$ 为真。

对于如果阀门 t_1 和 t_2 是打开的并且浴缸未被塞住，那么表示如何将水从 d_1 排水管排除的确定

子句公理化如下：

```

pressurized_p1
pressurized_p2 ← on_t1 ∧ pressurized_p1
flow_shower ← on_t2 ∧ pressurized_p2
wet_bath ← flow_shower
flow_d2 ← wet_bath ∧ unplugged_bath
flow_d1 ← flow_d2
on_t1
on_t2
unplugged_bath

```

(a) 完成对与公理化浴缸相同形式的水槽的公理化，并在 AILog 中测试。

(b) 如果水暖工没有在房子里，你希望用户为你提供哪些信息？改变该公理化系统，以便于询问用户有关的信息。

(c) 如果水槽溢出或者浴缸溢出，公理化地板如何为潮湿的。你需要以下条件：当塞子塞住并且水流入时，则它们会发生溢出。在你给出预期解释时，你可能需要引入新的原子命题。（假设塞子和阀门在一个小时内有相同的位置，你不需要去公理化如何动态打开阀门或插入、移除塞子）。在 AILog 中进行测试。

(d) 假设热水系统安装到了阀门 t_1 的左侧。在水管上有另一个阀门，控制热水来进行洗澡和进入水槽中（对于每一个水槽，有分开的冷水和热水阀门）。将这个信息加入你的公理化系统中，并对于所有你所使用的命题给出符号。在 AILog 中测试。

209

5.3 给定以下的知识库：

```

a ← b ∧ c
a ← e ∧ f
b ← d
b ← f ∧ h
c ← e
d ← h
e
f ← g
g ← c

```

(a) 给出该知识库的模型。

(b) 给出一个不是该知识库的模型的解释。

(c) 给出两个该知识库的逻辑推理的原子。

(d) 给出两个不是该知识库的逻辑推理的原子。

5.4 给定包含以下子句的知识库 KB：

```

a ← b ∧ c
b ← d
b ← e
c
d ← h
e
f ← g ∧ b
g ← c ∧ k
j ← a ∧ b

```

(a) 写出对于这个例子的自底向上验证过程。给出 KB 所有的逻辑结论。

(b) f 不是 KB 的逻辑结论。给出 KB 的一个模型，在该模型中 f 为假。

(c) a 是 KB 的逻辑结论, 对于查询问题 ask a , 给出一个自顶向下的推导。

- 5.5 自底向上验证过程可以包含一个用户询问机制, 该机制需要询问用户所有的原子命题。自底向上的验证程序如何在不用询问用户每一个可询问的原子的前提下, 保证对所有原子的验证都具有确定子句的知识库的逻辑完备性?

210

- 5.6 本题研究如何在拥有明确的语义的情况下调试程序。本书网站中的文件 `elect_bug2.ail` 是对于图 5-2 的电气布线领域的一种公理化的形式, 但是它包含了一个漏洞子句(即在图中的预期解释中有一个为假)。本习题的目标是通过使用 AILog 来找到该漏洞子句, 在 AILog 中使用例 5-5 所给定的符号定义。你不需要去看知识库就可以找到漏洞规则!(如果你喜欢, 你可以通过观察知识库来找到漏洞子句, 但是这不会在本题中帮助你。)所有你需要知道的是程序中符号的意义和在预期解释中哪些是真的。

查询 `lit_l1` 可以被证明, 但是在预期解释中它为假。使用 AILog 中的 `how` 问题来找到在预期解释中哪些子句的头部是假的, 哪些为真。这就是漏洞规则。

- 5.7 考虑下面用以解释什么人形迹可疑的知识库和假说:

```
goto_forest ← walking
get_gun ← hunting
goto_forest ← hunting
get_gun ← robbing
goto_bank ← robbing
goto_bank ← banking
fill_withdrawal_form ← banking
false ← banking ∧ robbing
false ← wearing_good_shoes ∧ goto_forest
assumable walking, hunting, robbing, banking
```

- (a) 假设观察到 `get_gun`, 对于这个观察来说所有的最小解释是什么?
 (b) 假设观察到 `get_gun ∧ goto_bank`。对于这个观察所有的最小解释是什么?
 (c) 在最小解释中, 有没有可以移除的观察? 为了对其进行解释, 必须要加入哪些条件?
 (d) `goto_bank` 的最小解释是什么?
 (e) `goto_bank ∧ get_gun ∧ fill_withdrawal_form` 的最小解释是什么?

- 5.8 假设对于一个指定的病人可能有 4 种可能的疾病: p 、 q 、 r 和 s 。 p 和 q 会导致斑点。 q 、 r 或者 s 其中的一个(或者多个)会导致发烧。病人已经长斑并且发烧。假设你已经决定根据症状使用溯因推理来诊断这个病人。

211

- (a) 请写出如何使用 Horn 子句和假设来表示这个知识。
 (b) 写出如何使用溯因推理来诊断这个病人。请明确写出查询和推理的结果。
 (c) 假设患者不能同时患有 p 和 s 。写出你在(a)问题中提出的知识库有哪些变动。写出如何使用溯因推理方法通过新的知识库来诊断病人。明确写出查询和推理的结果。

- 5.9 考虑以下的子句和完整性约束:

```
false ← a ∧ b
false ← c
a ← d
a ← e
b ← d
b ← g
b ← h
c ← h
```

假设假说是 $\{d, e, f, g, h, i\}$, 那么最小约束是什么?

- 5.10 深空一号(<http://nmp.jpl.nasa.gov/ds1/>)是由美国航空航天局在 1998 年 10 月推出的飞船。它采

用人工智能技术用于其判断和控制。详细的细节可以参考 Muscettola、Nayak、Pell 和 Williams [1998] 或者 http://ic.arc.nasa.gov/projects/remote_Agent/ (但这些资料对于完成本题并不是必需的)。

图 5-14 展示了 DS1 发动机设计中的一个部分。为了实现在发动机的推力, 必须注入燃料和氧化剂。为了确保其在多个故障(主要是阀门卡住或者失效)存在时仍能正常工作, 整个设计存在了大量的冗余。注意其值是黑或者白以及是否有阀门与这个任务无关。

每一个阀门可以是状态良好(或者失效)和可以被打开(或者不能被打开)。我们的任务目标是公理化这个问题, 这样我们要完成两个任务:

- (a) 在发动机中, 观察到了缺乏推动力, 而且观察给出了哪些阀门是打开的, 使用基于一致性的判断, 确定哪些可能是错误的。
- (b) 给定有推动力的目标, 给定某些阀门是状态良好的知识, 确定哪些阀门是应该被打开的。

对于其中的每一个问题, 必须考虑在知识库中的子句是什么, 假设是什么。

原子应该是以下形式的:

- 如果 V 是打开的, $open_V$ 为真。该原子为 $open_v1$, $open_v2$, 等等。
- 如果 V 正常工作, 则为 ok_V 为真。
- 如果 V 已经通过燃气进行了加压, 则 $pressurized_V$ 为真。你应该假设 $pressurized_t1$ 和 $pressurized_t2$ 为真。
- 如果发动机 E 已经进行了推动, 则 $thrust_E$ 为真。
- 如果在任何一个发动机中都没有推力存在, 则 $thrust$ 为真。
- 如果没有推力, 则 $nothrust$ 为真。

212

为了确保其为可控制的, 仅仅写出对于发动机 $e1$ 的输入规则。使用 AILog 通过一些实例测试你的代码。

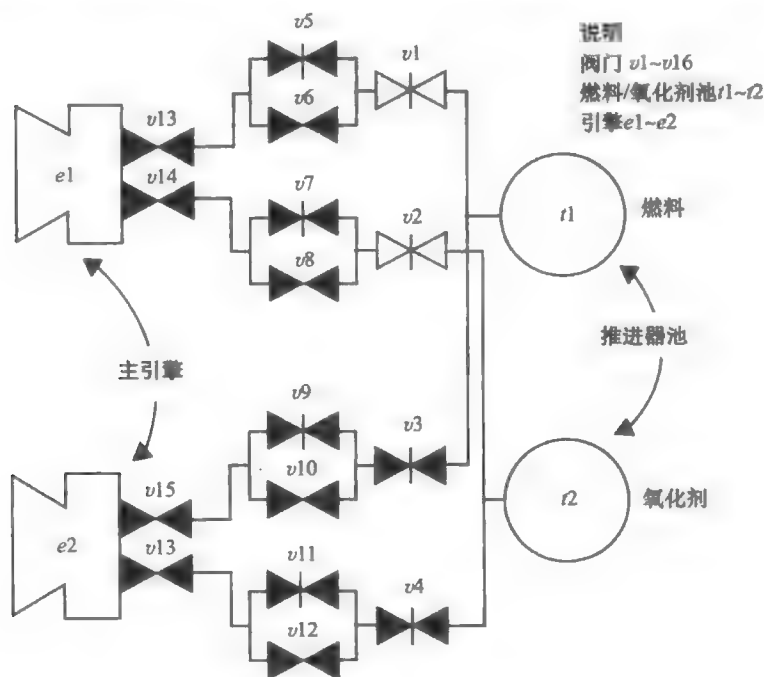


图 5-14 深空一号引擎设计

5.11 考虑在前面的问题中使用溯因判断方法。

假设如下:

- 阀门可以是打开或者关闭的, 对于其中的一些阀门, 我们知道它们是否是打开的, 对于另一些

我们不知道。

- 一个阀门可以是状态良好的,在这种情况下如果阀门是打开的燃气会流入,如果阀门是关闭的则不会流入;也可以是损坏的,在这种情况下燃气将永远不会流入;可以是卡住的,在这种情况下燃气是否流入与阀门是打开的还是关闭的无关;或者是泄漏的,在泄漏状态中,燃气将会泄露到阀门外面而不是流入。
- 有三种燃气传感器可以确定燃气是否泄漏(但不能确定是哪种燃气);第一个燃气传感器检测最右边的阀门(v_1, \dots, v_4)中的燃气,第二个燃气传感器检测中间的阀门(v_5, \dots, v_{12}),第三个燃气传感器检测最左边的阀门(v_{13}, \dots, v_{16})。

(a) 对这个领域知识进行公理化,以便系统可以解释引擎 e_1 是否进行推动,并且可以解释每一个传感器的气体状态。比如,可以解释为什么 e_1 在推动。可以解释为什么 e_1 没有推动,为什么通过第三个传感器有气体被检测到。

(b) 在一些非平凡的例子上测试你的公理化结果。

(c) 对于一些查询存在很多种解释。如何减少或者管理解释的数量使得溯因判断更加有效。

- 5.12 AllLog 中的 *askables* 是询问用户的原子, *assumables* 是在结果中选择的原子。假设你在公理化你家中的电路,且你有和例 5-3 相类似的公理化方法。现在你要为一个想要承租你的房子的新租客公理化该电路域,他可能想要确定在电路中有没有存在错误。

有一些原子是你可能知道的规则的约束,一些是该租客知道的,还有一些是你们两人都不知道的。将这些原子命题分为三个类别,指出哪些应该是可询问的,哪些是可假定的,并指出在你的划分下将会对结果产生哪些作用。

- 5.13 考虑在互联网上使用变化信息进行互动的采购 Agent 中,使用完整性约束和基于一致性诊断。为了解决某问题,该采购 Agent 将会针对事实询问一系列的问题,然而,信息资源有时是错误的。在用户获取矛盾信息时,如何自动确定哪些信息是错误的非常有用。

在本题中,我们考虑如何使用完整性约束和假设来确定不同的信息资源中显示出的错误信息。

在本题中,我们使用无意义的符号如 a, b, c, \dots , 但是在实际系统中,这些符号可能存在着关联性,比如 a 意味着“在夏威夷有滑雪板”,而 z 意味着“在夏威夷没有滑雪板”,或者 a 意味着“蝴蝶什么都不吃”,而 z 意味着“蝴蝶吃花蜜”。我们在本题中使用无意义的符号,因为计算机不需要获得其符号代表的意义,而是只需要将其作为无意义的符号来处理即可。

假设存在下面的信息资源和相关信息:

资源 s_1 : 资源 s_1 要求下面的子句为真:

$a \leftarrow h$

$d \leftarrow c$

资源 s_2 : 资源 s_2 要求下面的子句为真:

$e \leftarrow d$

$f \leftarrow k$

$z \leftarrow g$

j

资源 s_3 : 资源 s_3 要求下面的子句为真:

$h \leftarrow d$

资源 s_4 : 资源 s_4 要求下面的子句为真:

$a \leftarrow b \wedge e$

$b \leftarrow c$

资源 s_5 : 资源 s_5 要求下面的子句为真:

$g \leftarrow f \wedge j$

你自己: 假设你知道以下的子句为真:

$false \leftarrow a \wedge z$

c

k

由于它们在一起就产生了矛盾, 所以不是每一个资源都可信。

- (a) 通过用户在 AILog 中使用假设来编码知识。为了使用其中某一资源提供的子句, 你必须假设该资源是可信的。
- (b) 使用程序来找到哪些是可信资源的冲突(找到你可以确定为 *false* 的冲突)。
- (c) 假设你想要假定尽可能少的资源是不可信的。对于哪一个单一的资源, 如果其为不可信的, 将会导致矛盾?(假设其他的资源都是可信的。)
- (d) 哪两个资源会是导致矛盾的原因? 假设其他资源都是可信的, 在该情况下两个资源中的任何一个都不会导致矛盾。

- 5.14 假设你在某公司任职, 该公司建立了一种在线教学工具。由于你上过人工智能课程, 你的老板希望知道你基于考虑后对各种问题提出的意见。

他们计划为讲授基础物理(如力学和电磁学)建立一套智能教学系统。系统必须完成的一个任务是来判断学生可能出现的错误。

对于下面的陈述, 使用适当的语言回答每一个问题。如果回答的部分并没有被提问过或者只有一个问题被问到却给出了多于一个的答案时, 可能会惹恼你的老板。老板也不喜欢专业术语, 所以请使用简单的语言。

老板听说过基于一一致性的判断和溯因推理判断, 但是并不想知道它们在建立讲授基础物理智能教学系统中具体涉及的内容。

215

- (a) 解释在基于一一致性的判断中, 需要哪些关于物理和关于学生的知识。
- (b) 解释在溯因推理判断中, 需要哪些关于物理和关于学生的知识。
- (c) 在该领域中使用溯因推理判断比使用基于一一致性的诊断的优势有哪些?
- (d) 在该领域中使用基于一一致性的诊断比使用溯因推理判断的优势有哪些?

- 5.15 考虑图 5-11 中自底向上否定即失败验证程序。假设我们想要逐渐增加和删除子句。随着子句的增加, *C* 会如何变化? 随着子句的移除, *C* 会如何变化?

- 5.16 假设你在实现一个自底向上 Horn 子句解释程序, 并且你希望增加子句或者假说。当子句增加时, 最小解释将会受到何种影响? 当增加假说时最小解释会受到何种影响?

- 5.17 图 5-15 显示了一个简化了的无人航天器(*sc*)与地面控制中心(*gc*)之间的冗余通信网。有两种高带宽(高增益)的链接, 其通过中继卫星(*s1*, *s2*)与不同的地面天线(*a1*, *a2*)联系。此外, 也有一个直接的, 低带宽(低增益)的链接, 是地面控制中心的天线(*a3*)和飞船进行的联系。低增益链路受大气影响扰动, 如果飞船的低增益发射机(*sc_lg*)和地面天线 3 都是良好的, 且没有扰动(*no_dist*), 则其就进行工作。如果飞船的高增益发射器(*sc_hg*)、卫星天线(*s1_ant*, *s2_ant*)、卫星发射器(*s1_trans*, *s2_trans*)和地面天线(*a1*, *a2*)都是工作正常的, 则高增益链接会一直工作。

216

为了简单起见, 我们认为飞船仅仅通过这些渠道与地面控制中心传递信息。

以下知识库是我们所感兴趣的部分通信网络的形式化表示:

```

send_signal_lg_sc ← ok_sc_lg ∧ alive_sc
send_signal_hg_sc ← ok_sc_hg ∧ alive_sc
get_signal_s1 ← send_signal_hg_sc ∧ ok_s1_ant
get_signal_s2 ← send_signal_hg_sc ∧ ok_s2_ant
send_signal_s1 ← get_signal_s1 ∧ ok_s1_trans
send_signal_s2 ← get_signal_s2 ∧ ok_s2_trans
get_signal_gc ← send_signal_s1 ∧ ok_a1
get_signal_gc ← send_signal_s2 ∧ ok_a2
get_signal_gc ← send_signal_lg_sc ∧ ok_a3 ∧ no_dist

```

由于地面中心没有从飞船接收到信号(*no_signal_gc*), 地面中心非常紧张。可以明确地知道所有的地面天线是正常工作的(即 *ok_a1*、*ok_a2* 且 *ok_a3*), 并且知道卫星 *s1* 的发射器也是正常工作

的(*ok_s1_trans*)。但不能确定飞船、飞船发射器、卫星天线、*s2* 的发射器和大气扰动的状态。

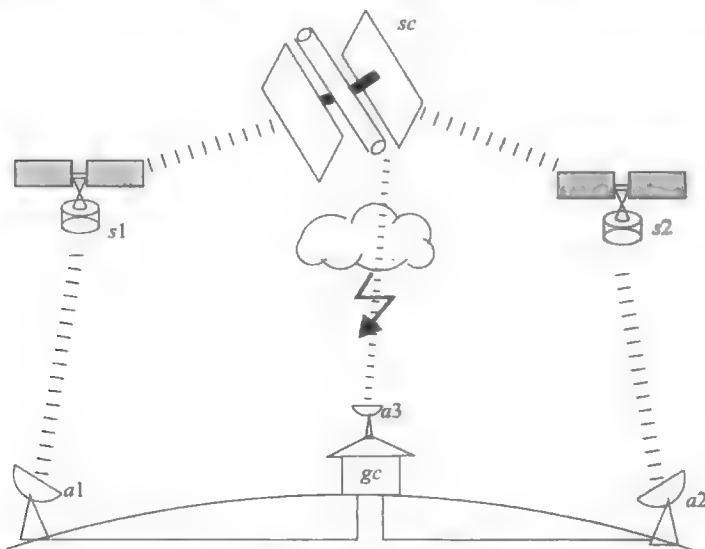


图 5-15 空间通信网络

- (a) 给出一组假说和一个完整性约束来为这个情况建模。
- (b) 使用(a)问题中得出的假说和完整性约束，哪些是最小约束？
- (c) 针对给定的情况，基于一致性的判断是什么？换句话说，哪些可能的违反假设的组合导致了地面中心不能接收到从飞船发来的信号？

5.18 (a) 解释为什么美国航空航天局希望针对习题 5.17 的领域问题，使用溯因推理而不是使用基于一致性的判断。

- (b) 假设大气扰动(*dist*)可能会在低带宽信号中导致静态信号或者无信号情况。为了获得静态信号，天线 *a3* 和飞船的低带宽发射器 *sc_lg* 都必须工作。如果 *a3* 或者 *sc_lg* 不工作或者 *sc* 已经失效，则没有信号。如果我们需要解释可能观察到的命题 *no_signal_gc*、*get_signal_gc*，或 *static_gc*，那么必须需要在习题 5.17 所示的知识库中加入什么规则和假设？你可以忽视高带宽链接。你可以创造任何你需要的符号。

不确定推理

值得注意的是，一门起源于对机会游戏考量的科学应该成为人类知识领域最重要的研究对象……生活中最重要的问题，绝大部分其实只是概率问题……

概率论归结到底不过是计算常识而已。

——Pierre Simon de Laplace[1812]

Agent 总是被迫在不完整信息的基础上做出决定，即便当 Agent 感知这个世界来试图找出更多信息时，它也很少发现世界的确切状态。就像机器人不会确切地知道目标所在，医生不会确切地知道患者究竟哪里有问题，老师不会确切地知道学生理解了什么。当 Agent 必须做出决定时，它们不得不使用他们所拥有的所有信息。本章主要考虑不确定性推理：通过对世界的观察来确定这个世界上什么是正确的。第 9 章中将这作为不确定性作用下的基础，在这种环境中，Agent 必须决定采取什么样的行动，即使它不能准确地预测其行动的结果。本章以概率开始，通过适当的独立性假设展示出怎样表示这个世界，并且说明怎样通过这些表示进行推理。

6.1 概率

为了做出一个好的决定，Agent 不能简单地假设这个世界是什么样子，更不能根据这些假设做出行动。它必须考虑多种可能的突发事件和它们的可能性。考虑下面的例子。

【例 6-1】 许多人认为在汽车行驶中系好安全带是明智的，因为在事故中，系安全带可降低严重伤害的危险。考虑一个会做出假设并根据这些假设做出决定的 Agent。如果 Agent 假设不会发生意外，它就不会为系安全带带来不便而烦恼。如果它假定会发生事故，它便不会出行。故而在这两种情况下都不会系好安全带！一个更加智能的 Agent 愿意系安全带，因为如果它发生意外，系安全带所造成的不便远不如它受伤或死亡增加的风险那样重要。它不会因为担心发生事故而不出门，因为移动远比基于极度谨慎而不出门带来更多的好处，即使会冒点发生事故的风险。决定是否走出去、是否系安全带取决于发生意外的可能性，在意外中系好安全带能带来多少帮助，系安全带带来的不便和走出去的重要性。不同的 Agent 可能做出不同的权衡。有些人不系安全带，有些人基于会发生意外的风险而选择不出去。

不确定性推理已经在概率论和决策理论领域有所研究，概率就是赌博(gambling)的运算。当 Agent 做出决定并且其行动的结果包含不确定性时，它就是在对结果进行赌博。然而，与在赌场的赌徒不同，Agent 不能选择退出决定不赌，无论它做什么涉及不确定性和风险(包括它什么都不做的情况)。如果它不考虑概率，它最终会在赌博中败给考虑概率的 Agent。当然，这并不意味着，做出最好的决定就能保赢。

我们中的许多人学习的概率论为抛硬币或是掷骰子，尽管这是一个提出概率论的好方法，但实际上概率可以用到比硬币和骰子丰富得多的应用中。在一般情况下，我们希望有一个用于信念的演算可以用来决策。

一种观点将概率看做是对信念的度量，而不是频率，为人们所熟知的是贝叶斯概率(bayesian probability)或主观概率(subjective probability)。主观这一术语不意味着任意，而是倾向于“主题”。例如，假设有三个 Agent，Alice、Bob 和 Chris，它们投掷一个骰子。假设 Alice 观察的结果是“6”，并告诉 Bob 结果是偶数，而 Chris 对结果一无所知。在这种情况下，Alice 对结果是“6”的概率为 1，Bob 对结果是“6”的概率为 $1/3$ （假设 Bob 相信 Alice，并且认为所有偶数的概率是相同的），Chris 的概率为 $1/6$ 。由于具有不同的知识导致它们的概率不同，这个概率是此次掷骰子的特定结果，而不是掷骰子的某些通用事件。对于其他投硬币事件的结果，这些 Agent 可以具有相同或不同的概率。

220

另一种是频率论(frequentist)观点，概率是长期运行重复事件的频率。贝叶斯概率适合于智能 Agent，因为在特定情况下信念度量是决策所需要的。Agent 不会遇到一般事件，但必须基于它们遇到的特殊情况的不确定性做出决定。

概率论可以被定义为知识如何影响信念的研究。对某个命题的相信程度 α ，可以利用 0 和 1 之间的数来衡量。概率 α 为 0 表示命题被认为肯定是假的（没有新的证据可以动摇这种信念），概率为 1 意味着命题被认为肯定是正确的。使用 0 和 1 纯粹是一个惯例。

采用概率的信念观点并不意味着统计数据将被忽略。对过去发生的事件所产生的统计数据是有重要影响力的知识，并且可以用于更新信念（请参阅第 7 章如何学习概率）。

我们假设的不确定性是认识论(epistemological)上的，与 Agent 对世界的认识有关，而不是本体论(ontological)的，即世界是怎样的。我们假设 Agent 关于命题真实性的知识是不确定的，没有假设真实的程度。例如，如果有人告诉你某个人非常高，你知道他有一定的高度；你对于他的真实高度只是有一个模糊的概念。

如果 Agent 的某个 α 的概率是大于 0 且小于 1 的，这并不意味着在某种程度上 α 为真，而是 Agent 不知道 α 是真还是假。概率反映出 Agent 无知的部分。

在本章其余的部分，我们忽略模拟的拥有概率的 Agent 本身，只考虑概率。

6.1.1 概率的语义

概率论是建立在世界和变量作为约束满足的相同的基础上的（见 4.2 节）。概率度量可能的世界，不带有去除某些世界并把每个其他的世界看成是可能世界的某些约束。概率论中涉及的变量为随机变量(random variable)，随机变量这一术语有点用词不当，因为它既不是随机的也不是变量。正如 4.2 节所讨论的，变量可以描述世界，一个世界对应于对每个变量的一种赋值。反过来，变量也可以由世界来描述，一个变量是一个函数，该函数对每个世界返回一个值。

首先，我们定义概率为世界集合的一种度量，然后定义概率是关于命题的，最后才是关于变量的。

世界的概率度量(probability measure)是从世界集合到非负实数集的一个函数 μ ，使得

- 如果 Ω_1 和 Ω_2 是世界的不相交集（即如果 $\Omega_1 \cap \Omega_2 = \{\}$ ），那么 $\mu(\Omega_1 \cup \Omega_2) = \mu(\Omega_1) + \mu(\Omega_2)$ ；
- 如果 Ω 是所有世界的集合， $\mu(\Omega) = 1$ 。

221

需要注意的是，只是习惯上使用 1 作为全集的概率，同样也可以使用 100。

当一些变量有无穷的定义域或是当无穷多个变量存在时，存在无穷多个世界是可能的。当存在无穷多个世界时，我们并不需要度量 Ω 的所有子集——只需度量可以用某种语言描述的集合，假设这种语言允许我们描述交集、并集、补集。通过这些操作的所描述的

子集的集合具有数学家称之为代数(algebra)的结构。

对于世界 ω 来说,可以通过定义 $\mu(\omega)=\mu(\{\omega\})$ 把度量 μ 扩展到世界上。当存在有限多的世界时,个别世界的度量足以定义 μ 。当存在无限多的世界时,可能个别世界的度量不足以定义 μ ,或者说对个别世界进行度量是没有意义的。

【例 6-2】假设世界对应一个特定人的可能实值高度,以厘米为单位。在这个例子中,有无限多可能的世界。高度集合在范围 $[175, 180)$ 内的概率为 0.2,在范围 $[180, 190)$ 内的概率为 0.3。那么在范围 $[175, 190)$ 内的概率为 0.5。然而,任何特定高度的度量可以 \blacktriangleleft 为零。

如 5.1 节中描述的,原始命题是将一个值赋给变量。命题是由原始命题通过使用逻辑连接词连接而成。下面我们就利用这个性质来定义变量的概率分布。

命题 α 的概率记作 $P(\alpha)$,是可能世界集中使 α 为真的集合的度量。即

$$P(\alpha)=\mu(\{\omega:\omega\models\alpha\})$$

其中, $\omega\models\alpha$ 代表在世界 ω 中 α 为真。因此, $P(\alpha)$ 是世界集中 α 为真的度量。

这里使用的符号 \models 不同于前面章节中使用的(见 5.1.2 节)。在前面章节中,符号的左侧表示的是一个知识库;这里,左侧表示一个世界。其具体的含义从上下文来看应该是明确的。

随机变量 X 上的概率分布(probability distribution) $P(X)$,是一个将 X 域映射为实数的函数,使得给出一个值 $x\in\text{dom}(X)$, $P(x)$ 是命题 $X=x$ 的概率。我们也可以类似地定义一组变量的概率分布。例如, $P(X,Y)$ 是 X 和 Y 上的概率分布,即 $P(X=x,Y=y)$,其中的 $x\in\text{dom}(X)$, $y\in\text{dom}(Y)$,它有值 $P(X=x\wedge Y=y)$,其中 $X=x\wedge Y=y$ 是一个命题, P 是命题的函数。当把概率集合看成是一个整体时,我们将使用概率分布。

222

6.1.2 概率公理

前一节给出了概率的语义定义。我们也可以给出一个概率的公理化定义,指出在信念演算中我们可能需要的属性公理。

概率密度函数

当域连续时(例如,一个实数子集),概率有时根据概率密度函数来定义。概率密度函数提供了一种度量可能世界集合的方式。这种度量根据概率密度函数的积分来定义,积分正式地定义为离散化越来越细时它的极限值。

在这本书中我们唯一使用的非离散概率分布的域是实数线。在这种情况下,每个实数都对应一个可能世界。概率密度函数(probability density function)我们记为 p ,它是一个从实数到非负实数积分为 1 的函数。一个值在 a 和 b 之间的实值随机变量 X 的概率给出如下:

$$P(a\leq X\leq b)=\int_a^b p(X)dX$$

参数分布(parametric distribution)是一个其密度函数可以由公式来描述的分布。虽然并非所有的分布都可以用公式描述,但是所有我们能够表示的分布都可以。有时统计学家使用术语参数化,代表那些可以由一组固定的、有限数量的参数来描述的分布。非参数化分布(non-parametric distribution)的参数数目是不固定的。(奇妙的是,非参数化通常意味

着“许多参数”。)

一个常见的参数分布是均值为 μ , 方差为 σ^2 的正态分布或高斯分布(normal or Gaussian distribution)。定义为

$$p(X) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}((X-\mu)/\sigma)^2}$$

其中, σ 是标准偏差。用正态分布测量误差, 此处 μ 为平均值, σ 为值的偏差。由拉普拉斯[1812]证明的**中心极限定理**(central limit theorem)表明, 独立误差的总和将逼近高斯分布。这和其他优异的数学特点使得正态分布被广泛使用。

其他分布, 包括贝塔和狄利克雷分布, 将在不确定性学习一节中讨论。

假设 P 是一个将命题转化为实数的函数, 满足以下三个**概率公理**(axiom of probability):

公理 1 对于任意命题 α , $0 \leq P(\alpha)$ 。也就是说, 任何命题的概率不能为负数。

公理 2 如果 τ 为重言式, 则 $P(\tau)=1$ 。也就是说, 如果 τ 对所有可能的世界均为真, 则其概率为 1。

公理 3 当 α 和 β 互为对立命题时, 也就是说, 当 $\neg(\alpha \wedge \beta)$ 为重言式时, $P(\alpha \vee \beta) = P(\alpha) + P(\beta)$ 。即如果两个命题不能同时为真(它们互斥), 其析取的概率是它们的概率之和。

这些公理具有直观属性, 使我们获得合理的信念度量。如果信念度量遵从这些直观的公理, 则不论它是否是实际频率计数, 都符合概率论。这些公理形成一个合理而完备的公理化概率意义, 合理性是指这些公理被用可能世界语义定义的概率遵循, 完备性是指任何遵循这些公理的信念系统都具有概率语义。

命题 6.1 如果存在有限数目的有限离散随机变量, 公理 1、2、3 关于语义是合理的和完备的。

这些公理的语义真实性是很容易检查的。换句话说, 你可以使用这些公理来从世界概率计算任何概率, 因为两个世界中的描述是互斥的。其完全的证明留给读者作为练习。

命题 6.2 对于所有命题 α 和 β , 以下都成立:

(a) 否定命题:

$$P(\neg\alpha) = 1 - P(\alpha)$$

(b) 如果 $\alpha \leftrightarrow \beta$, 则 $P(\alpha) = P(\beta)$ 。也就是说, 逻辑上等价的命题具有相同的概率。

(c) 情况推理:

$$P(\alpha) = P(\alpha \wedge \beta) + P(\alpha \wedge \neg\beta)$$

(d) 如果 V 是域 D 中的一个随机变量, 那么对于所有命题 α ,

$$P(\alpha) = \sum_{d \in D} P(\alpha \wedge V = d)$$

(e) 非互斥命题的析取:

$$P(\alpha \vee \beta) = P(\alpha) + P(\beta) - P(\alpha \wedge \beta)$$

证明:

(a) 命题 $\alpha \vee \neg\alpha$ 和 $\neg(\alpha \wedge \neg\alpha)$ 是重言式。因此, $1 = P(\alpha \vee \neg\alpha) = P(\alpha) + P(\neg\alpha)$ 。重新排列给出了所需的结果。

(b) 如果 $\alpha \leftrightarrow \beta$, 则 $\alpha \vee \neg\beta$ 是重言式, 所以 $P(\alpha \vee \neg\beta) = 1$ 。 α 和 $\neg\beta$ 是矛盾命题, 所以可以由公理 3 给出 $P(\alpha \vee \neg\beta) = P(\alpha) + P(\neg\beta)$ 。由 (a) 有 $P(\neg\beta) = 1 - P(\beta)$ 。因此, $P(\alpha) + 1 -$

$P(\beta)=1, P(\alpha)=P(\beta)$ 。

(c) 命题 $\alpha \leftrightarrow ((\alpha \wedge \beta) \vee (\alpha \wedge \neg \beta))$ 和 $\neg((\alpha \wedge \beta) \wedge (\alpha \wedge \neg \beta))$ 是重言式。因此, $P(\alpha) = P((\alpha \wedge \beta) \vee (\alpha \wedge \neg \beta)) = P(\alpha \wedge \beta) + P(\alpha \wedge \neg \beta)$ 。

(d) 证明类似于(c)。

(e) $(\alpha \vee \beta) \leftrightarrow ((\alpha \vee \neg \beta) \vee \beta)$ 为重言式。因此,

$P(\alpha \vee \beta) = P((\alpha \vee \neg \beta) \vee \beta) = P(\alpha \vee \neg \beta) + P(\beta)$

由(c)有, $P(\alpha \wedge \neg \beta) = P(\alpha) - P(\alpha \wedge \beta)$ 。因此,

$P(\alpha \vee \beta) = P(\alpha) - P(\alpha \wedge \beta) + P(\beta)$ ■

6.1.3 条件概率

通常情况下, 我们不仅想知道一些命题的先验概率, 而且想知道当 Agent 观察到新证据时如何更新信念。

基于命题 e 的命题 h 的信念度量称作给定 e 下 h 的**条件概率**(conditional probability), 记为 $P(h|e)$ 。

公式 e 代表 Agent 对世界所有**观察**(observation)的合取, 称为**证据**(evidence)。已知证据 e , 条件概率 $P(h|e)$ 是 h 的**后验概率**(posterior probability)。概率 $P(h)$ 是 h 的**先验概率**(prior probability), 与 $P(h|true)$ 相同, 因为它是 Agent 在观察到任何事物之前的概率。

后验概率以 Agent 知道的特定情况下的一切为条件, 所有的证据都必须以获得正确的后验概率为条件。

【例 6-3】 对于诊断助手来说, 患者的症状就是证据。诊断 Agent 在了解特定病人之前使用可能疾病的先验概率分布。Agent 在获得一些证据之后会使用后验概率。当 Agent 通过与患者进行讨论, 观察症状, 或得到实验室测试的结果后获取新的证据, 它必须更新它的后验概率以反映新的证据。新证据是旧证据和新观察的结合体。 ◀

【例 6-4】 机器人从其传感器接收到的信息是它的证据。当传感器有噪声时, 证据是那些已知的东西, 诸如由传感器接收到的特定模式, 而不是有一个人站在机器人面前。机器人可能对世界有误解, 但是它知道收到了什么样的信息。 ▶

225

1. 条件概率的语义

证据 e 将排除所有可能与其不匹配的世界。就像逻辑结论的定义, 给定的公式 e 会选择使其为真的可能世界。证据 e 在所有可能世界上推导出一个新的度量 μ_e , 当所有 e 为假时, 可能世界的度量为 0, 其余的世界进行标准化, 所以世界的度量总和为 1。

在这里, 我们回顾定义条件概率的基本原则。在面对不寻常的情况时, 这个基本定义通常是很有用的。

度量的定义遵循以下两个直观的属性:

- 如果 S 是一个可能世界的集合, 对于某个常数 c (下面推导), 所有使 e 为真的度量, 定义为 $\mu_e(S) = c \times \mu(S)$ 。
- 如果 S 是一个世界集合, 所有使 e 为假的度量定义为 $\mu_e(S) = 0$ 。

其他可能的信念度量

证明其他信念度量具有不确定性。例如, 信念 $\alpha \wedge \beta$ 是信念 α 和信念 β 的某种函数, 像这样的信念度量叫做**组合式**(compositional)**度量**。考虑单一硬币的投掷就可以看出为什么这种度量是不明智的。比较一下两种情况, 第一种情况, α_1 代表投掷硬币正面朝上并且 β

代表投掷硬币背面朝上, 第二种情况, α_2 代表投掷硬币正面朝上并且 β_2 代表投掷硬币正面朝上。对于这两种情况来说, 信念 α_1 似乎与信念 α_2 , 信念 β_1 与信念 β_2 相等。但是信念 $\alpha_1 \wedge \beta_1$ (其实是不存在的) 不同于信念 $\alpha_2 \wedge \beta_2$ (与 α_2 相等)。

条件概率 $P(f|e)$ 不同于蕴含式 $P(e \rightarrow f)$ 的概率。后者与 $P(\neg e \vee f)$ 相同, 即 f 为真或者 e 为假所解释的度量。例如, 假设有一个域, 鸟很稀少, 且其中的一小部分为不飞的鸟。这里 $P(\neg flies|bird)$ 代表那些不飞的鸟的比例, 通常非常低。 $P(bird \rightarrow \neg flies)$ 与 $P(\neg bird \vee \neg flies)$ 相同, 由非鸟类占主导地位, 所以会很高。类似的, 概率 $P(bird \rightarrow flies)$ 也很高, 由非鸟类占主导地位。很难设想一个蕴含式的概率是一种适当的或是有用的知识。

226

我们令 μ_e 为一概率度量, 所以如果 Ω 是所有可能世界的集合, $\mu_e(\Omega) = 1$ 。因此, $1 = \mu_e(\Omega) = \mu_e(\{\omega : \omega \models e\}) + \mu_e(\{\omega : \omega \not\models e\}) = c \times \mu_e(\{\omega : \omega \models e\}) + 0 = c \times P(e)$, 因此 $c = 1/P(e)$ 。

已知证据 e 下 h 的条件概率用 μ_e 来表示, 其可能世界中 h 为真, 即

$$\begin{aligned} P(h|e) &= \mu_e(\{\omega : \omega \models h\}) = \mu_e(\{\omega : \omega \models h \wedge e\}) + \mu_e(\{\omega : \omega \models h \wedge \neg e\}) \\ &= \frac{\mu_e(\{\omega : \omega \models h \wedge e\})}{P(e)} + 0 = \frac{P(h \wedge e)}{P(e)} \end{aligned}$$

上面所给出的最后的形式通常作为条件概率的定义。

本章的其余部分假设, 如果 e 为证据, $P(e) > 0$ 。我们不考虑那些命题概率为 0 的情况(比如, 世界集合的度量为 0)。

条件概率分布(conditional probability distribution)为变量函数, 记为 $P(X|Y)$, 其中 X 和 Y 是变量或变量集; 对于 X , 给定一个值 $x \in \text{dom}(X)$, 对于 Y , 给定一个值 $y \in \text{dom}(Y)$, 其给出 $P(X=x|Y=y)$ 的值, 后者是命题的条件概率。

条件概率的定义可以让我们将一个合取分解为几个条件概率的乘积:

命题 6.3 (链式法则) 条件概率可以用来分解合取。对于任意命题 $\alpha_1, \dots, \alpha_n$:

$$\begin{aligned} P(\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_n) &= P(\alpha_1) \times \\ &\quad P(\alpha_2 | \alpha_1) \times \\ &\quad P(\alpha_3 | \alpha_1 \wedge \alpha_2) \times \\ &\quad \vdots \\ &\quad P(\alpha_n | \alpha_1 \wedge \dots \wedge \alpha_{n-1}) \\ &= \prod_{i=1}^n P(\alpha_i | \alpha_1 \wedge \dots \wedge \alpha_{i-1}) \end{aligned}$$

这里如果乘积中任意项为 0, 则假定上式中右侧为 0(即使其中有些是未定义的)。

请注意, 如果为每个概率增加了相同的证据, 则任何有关非条件概率的定理也是有关条件概率的定理。这是因为条件概率度量是另一种概率度量。

2. 贝叶斯规则

当 Agent 观察到新的证据时, 它必须更新其概率。一条新证据与旧证据相结合形成一个完整的证据集。

227

背景知识与观察

背景知识和观察之间的不同之处在 5.3.1 节中描述过。当我们使用不确定性推理时, 其背景模型是根据概率模型来描述的, 观察形成条件性证据。

在概率中有两种方法来说明 a 为真:

- 第一种, a 的概率为 1, 即写为 $P(a)=1$ 。
- 第二种, 对 a 进行条件限制, 把 a 放在条件栏的右侧, 即 $P(\cdot|a)$ 。

第一种方法指出, a 在所有可能世界中为真。第二种方法指出, Agent 只对 a 偶然为真的世界感兴趣。

假设 Agent 知道一种特定的动物:

$$P(\text{flies}|\text{bird})=0.8$$

$$P(\text{bird}|\text{emu})=1.0$$

$$P(\text{flies}|\text{emu})=0.001$$

如果它断定这种动物是鸕鹚(*emu*, 一种不会飞的鸟), 则不能增加语句 $P(\text{emu})=1$ 。没有概率分布能够满足这四项要求。如果在所有可能世界中 *emu* 为真, 就不会出现概率为 0.8, 即有个别会飞的鸟的情况。相反, Agent 必须把条件限制在事实上, 即 *emu* 才是个别情况。

为了建立一个概率模型, 知识库的设计者必须考虑到一些知识, 并在此知识基础上来建立概率模型, 后续所获得的知识必须看成是有条件的观察。

假设 Agent 在一段时间内所观察到的知识是由命题 k 给出的。Agent 的后续的信念状态可以由下面任意一项来模拟:

- 在 Agent 观察 k 之前, 基于测度 μ 构建一种概率论, 然后以证据 k 为条件, 与后续证据 e 相结合。
- 基于测度 μ_k 构建一种概率论, 来模拟 Agent 观察 k 之后的信念, 然后以后续证据 e 作为条件。

无论使用哪种构建, 所有的后续概率都是完全相同的。有时直接构建 μ_k 很简单, 因为其模型不用必须满足 k 为假的情况。然而, 有时构建 μ 和条件限制 k 相对简单。

重要的是, 概率模型是合理的和每一个后续观察是有条件的, 两者是连贯的。

228

贝叶斯规则指明 Agent 应该怎样基于一条新的证据来更新命题的概率。

假设 Agent 基于观察证据 k 的命题 h 的当前概率为 $P(h|k)$, 随后观察到 e 。对 h 的新概率为 $P(h|e \wedge k)$ 。贝叶斯规则告诉我们, 随着新证据的获得 Agent 是如何更新假定 h 的概率的。

命题 6.4(贝叶斯规则) 只要 $P(e|k) \neq 0$,

$$P(h|e \wedge k) = \frac{P(e|h \wedge k) \times P(h|k)}{P(e|k)}$$

通常隐含背景知识 k 。在这种情况下, 如果 $P(e) \neq 0$, 则

$$P(h|e) = \frac{P(e|h) \times P(h)}{P(e)}$$

$P(e|h)$ 是假设 h 的**可能性**(likelihood), $P(h)$ 为假设 h 的**前提**(prior)。贝叶斯规则表明, 后验概率与可能性和先验概率的乘积成正比。

证明: 合取的交换率是指 $h \wedge e$ 与 $e \wedge h$ 相等, 对于给定的 k 它们具有相同的概率。有两种不同的方式使用乘法规则:

$$P(h \wedge e|k) = P(h|e \wedge k) \times P(e|k) = P(e|h \wedge k) \times P(h|k)$$

定理通过 $P(e|k)$ 划分右侧部分。

通常情况下, 贝叶斯规则是用来比较各种假设(h_i)的, 可以注意到, 分母 $P(e|k)$ 是常数, 不依赖于特定的假设。当比较假设的相对后验概率时, 可以忽略分母。为了获得后验概率, 分母可以通过情况推理来计算。如果 H 是独立的, 并且代表了所有可能假设所

覆盖的命题的集合, 那么

$$P(e|k) = \sum_{h \in H} P(e \wedge h|k) = \sum_{h \in H} P(e|h \wedge k) \times P(h|k)$$

因此, 贝叶斯规则的分母是通过总和所有假设的分子而获得的。当假设空间很大时, 其分母是难以计算的。

一般情况下, 估计 $P(e|h \wedge k)$ 或 $P(h|e \wedge k)$ 中的一个会比另一个简单得多。当一个域中存在因果关系的理论时这种情况时常发生, 并且不同假设的预测——每一个假说 h_i 的 $P(e|h_i \wedge k)$ ——可以由域理论来推导。

【例 6-5】 假设诊断助手对图 1-8 的开关 s_i 的诊断感兴趣。我们期望这个模型能够指明开关的输出关于其输入、开关的位置以及开关的状态是怎样一个函数(可以是工作, 短路, 安装上下颠倒, 等等)。贝叶斯规则可以让 Agent 根据给定的其他信息推断出开关的状态。

【例 6-6】 假设 Agent 具有火警可靠性的信息。如果发生火灾它会知道警报的可能性是多大。如果有警报, 它必须要知道发生火灾的概率, 它可以使用贝叶斯规则:

$$P(\text{fire}|\text{alarm}) = \frac{P(\text{alarm}|\text{fire}) \times P(\text{fire})}{P(\text{alarm})}$$

假设发生火灾, 其 $P(\text{alarm}|\text{fire})$ 是报警的概率。它是对警报可靠性的度量。 $P(\text{fire})$ 是在没有其他信息的情况下发生火灾的概率。它是对建筑易发生火灾的度量。 $P(\text{alarm})$ 是在没有其他信息的情况下报警的概率。

6.1.4 期望值

可以使用概率来给出任何数值随机变量的期望值(如, 一个域是实数的一个子集)。变量的期望值是变量的加权平均值, 其在每个可能世界的值由可能世界的度量来加权。

假设 V 是一个随机变量, 它的域为数值域, ω 是一个可能世界。定义 $V(\omega)$ 为域 V 中的值 v , 使得 $\omega \models V = v$, 也就是说, 我们把一个随机变量作为世界的函数。

数值变量 V 的期望值(expected value), 记为 $\epsilon(V)$:

$$\epsilon(V) = \sum_{\omega \in \Omega} V(\omega) \times \mu(\omega)$$

这里指存在有限多的世界时。当存在无限多的世界时, 我们必须积分。

【例 6-7】 如果坏开关的数量是一个整数值的随机变量,

$\epsilon(\text{number_of_broken_switches})$

会给出坏开关的期望数。如果世界根据概率模型运行, 将给出坏开关长期运行的平均数。

用类似的语义定义条件概率, 以证据 e 为条件变量 X 的条件期望值(conditional expected value)记为 $\epsilon(V|e)$:

$$\epsilon(V|e) = \sum_{\omega \in \Omega} V(\omega) \times \mu_e(\omega)$$

因此,

$$\epsilon(V|e) = \frac{1}{P(e)} \sum_{\omega \models e} V(\omega) \times P(\omega) = \sum_{\omega \in \Omega} V(\omega) \times P(\omega|e)$$

【例 6-8】 假设灯 l_1 不亮, 坏开关的期望数如下:

$\epsilon(\text{number_of_broken_switches} | \neg \text{lit}(l_1))$

这是通过所有灯 l_1 不亮的所有世界中坏开关的平均数目获得的。

6.1.5 信息理论

概率形成信息理论的基础。在本节中,我们将简要介绍信息理论。

位(bit)是一个二进制数。因为位有两个可能的值,可以用它来区分两个项目。通常,这两个值写作0和1,但它们也可以写为任意两个不同的值。

2个位可以区分4个项目,分别记为00、01、10和11。同样,3个位可以区分8个项目。在一般情况下, n 个位可以区分 2^n 个项目。因此,我们可以用 $\log_2 n$ 位来区分 n 个项目。这可能是令人惊讶的,但我们可以通过把概率问题考虑进去从而做得更好。

【例6-9】假设你想设计一个编码来区分集合 $\{a, b, c, d\}$ 中的元素,其中 $P(a)=1/2$, $P(b)=1/4$, $P(c)=1/8$, $P(d)=1/8$ 。考虑下面的代码:

```
a 0    c 110
b 10   d 111
```

这种编码,有时会使用1位,有时会使用3位。平均来说,它使用

$$P(a) \times 1 + P(b) \times 2 + P(c) \times 3 + P(d) \times 3 = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{3}{8} = 1 \frac{3}{4} \text{ 位}$$

例如,8字符的字符串aacabbda的编码为14位00110010101110。

231

这个编码需要用 $-\log_2 P(a)=1$ 位来区分 a 不同于其他符号。要区分 b ,需要用 $-\log_2 P(b)=2$ 位。为了区分 c ,需要 $-\log_2 P(c)=3$ 位。

构建一种编码来识别 x 可能需要 $-\log_2 P(x)$ 位(或比这更大的整数,如果 x 是唯一需要发送的)。假设想要传送或储存一个符号序列,并且知道这些符号的概率分布。符号 x 的概率 $P(x)$ 需要 $-\log_2 P(x)$ 位。为了发送一个序列,每个符号平均需要

$$\sum_i -P(x) \times \log_2 P(x)$$

位来发送。这个值仅仅依赖于这些符号的概率分布,这就是所谓的分布信息内容(information content)或熵(entropy)。

类似于条件概率,它所需要的描述给定证据 e 的分布的位期望数为

$$I(e) = \sum_i -P(x|e) \times \log_2 P(x|e)$$

如果存在一个测试可以区分 α 为真和 α 为假的情况,此测试的信息增益(information gain)为

$$I(\text{true}) - (P(\alpha) \times I(\alpha) + P(\neg\alpha) \times I(\neg\alpha))$$

其中 $I(\text{true})$ 为在测试之前需要位的期望数, $P(\alpha) \times I(\alpha) + P(\neg\alpha) \times I(\neg\alpha)$ 为测试之后位的期望数。

在以后的章节中,我们将对任务数使用信息的概念。

- 在诊断中, Agent 可以选择一个提供最多信息的测试。
- 在决策树学习中,信息理论提供了一个有用的标准用来选择属性划分:划分能够提供最大信息增益的属性。它要区分的元素在目标概念中取不同的值,其概率是由保存在每个节点的训练集中的每个值的比例来获得的。
- 在贝叶斯学习中,信息理论在已知一些数据时,为决定最好的模型提供了基础。

6.2 独立性

概率的公理是非常弱的,在允许的条件概率上提供的约束很少。例如,如果有 n 个二进制变量,就要从任意得出的条件概率中为一个完整的概率分布分配 2^n-1 个数。为了确

232 定任意概率,可能需要启动一个庞大条件概率或可能世界概率的数据库。

有两种主要的方法可以克服对于庞大数目的需求:

独立性 假设一个命题 Y 的真实性知识在其他命题 Z 的情况下不会影响到命题 X 的概率。我们说 X 在给定 Z 的情况下独立于 Y , 定义在下文中给出。

最大熵或随机世界 没有其他的知识, 假设一切都是尽可能随机的。也就是说, 概率分布尽可能与有用信息的分布相一致。

这里详尽地论述上述的第一部分(请参阅下面描述的“减少数目”的内容)。

只要 $P(h|e)$ 的值不为 0 或 1, $P(h|e)$ 的值就不能限制 $P(h|f \wedge e)$ 的值。后者的概率可以在 $[0, 1]$ 内取任意值: 如果 f 蕴含 h , 它是 1, 如果 f 蕴含 $\neg h$, 它是 0。

依据概率论, 没有理由说为什么加拿大女王的名字不应该与决定灯是否亮所采纳的开关位置同等重要。然而, 领域知识可能告诉我们这是不相干的。

在本节中, 我们提出了一种表示, 使我们可以模拟世界结构, 当指定概率时相关命题是局部的, 非直接相关变量可以被忽略。开发这种结构可以进行高效的推理。

一种常见的定性知识是 $P(h|e) = P(h|f \wedge e)$ 的形式。这个等式表示 f 与给定 e 下的 h 的概率不相关。例如, 伊丽莎白是加拿大的女王与开关 s_1 关闭时 ω_2 带电的概率不相关。这种思想适用于随机变量, 正如下面的定义:

如果对于所有的 $x \in \text{dom}(X)$, $y \in \text{dom}(Y)$, $y' \in \text{dom}(Y)$, $z \in \text{dom}(Z)$, 随机变量 X 有**条件地独立**(conditionally independent)于给定随机变量 Z 下的随机变量 Y , 使得 $P(Y=y \wedge Z=z) > 0$, $P(Y=y' \wedge Z=z) > 0$, $P(X=x | Y=y \wedge Z=z) = P(X=x | Y=y' \wedge Z=z)$, 即给定一个 Z 值, 知道 Y 的值不会影响到对 X 值的信念。

命题 6.5 只要定义好条件概率, 下列 4 个陈述是等价的:

- 1) X 在有条件限制下独立于给定 Z 下的 Y 。
- 2) Y 在有条件限制下独立于给定 Z 下的 X 。
- 3) $P(X|Y, Z) = P(X|Z)$ 。也就是说, 在上下文中给定一个 Z 值, 如果给定一个 Y 值, 将在 X 中得到相同的概率, 与没有给定 Y 值一样。
- 4) $P(X, Y|Z) = P(X|Z)P(Y|Z)$ 。

证明留作练习。(见习题 6.1)

减少数目

允许独立性表示和使用最大熵或随机世界之间的区别, 突出了知识表示观点的重要不同:

- 第一种观点, 知识表示提供了一种高层次的建模语言, 可以让我们用一种合理自然的语言构建一个域。根据这一观点, 希望知识表示设计者规定怎样使用知识表示语言, 希望他们能够提供一份描述兴趣领域的用户说明书。
- 第二种观点, 知识表示应该允许添加有关一个域的任何知识, 知识表示应以常识的方式来填充。根据这一观点, 知识表示设计者指定特定知识进行编码是不合理的。

用错误的标准来判断知识表示不会产生一个公正的评价。

对于一个特定的独立变量来说, 信念网络是一种表示, 信念网络应该被看做一种建模语言。通过信念网络简洁表达出的独立性可以简明自然地表示许多领域。这并不意味着我们仅投入大量的事实(或概率)就能期待一个合理的答案, 必须考虑域、涉及的变量以及变

量之间呈现怎样的依赖关系。当用这个标准来评判时,信念网络就形成了一个有用的表示规划。

一旦定义信念网络的网络结构和变量域,就确切地给出所需的数目(条件概率)。用户不能只是简单地添加任意的条件概率,必须遵循网络的结构。如果提供的信念网络所需的数字与本地一致,整个网络将是一致的。

相反,最大熵或随机世界方法可以推断出绝大部分与概率知识库相一致的随机世界,它们形成了第二类的概率知识表示。对于随机世界方法,任何恰巧可用的数字都可以增加和使用,但是,如果允许增加任意的概率,知识就不易与概率公理相一致。此外,如果假设不明确就很难证明答案的正确性。

234

如果 $P(X, Y) = P(X)P(Y)$, 则变量 X 和 Y 是无条件独立(unconditionally independent)的,也就是说,如果不给出观察,它们是条件独立的。注意, X 和 Y 无条件独立并不意味着在给定其他信息 Z 时它们就是有条件独立的。

条件独立对一个域来说是一个有用的假设,其域通常会自然评定出并且可以给出有用的表示。

6.3 信念网络

条件独立的概念可以给出许多域的简洁表示。思想是,给定一个随机变量 X , 在 X 条件独立于给定直接影响变量值的其他变量情况下,也许存在一个变量的小集合直接影响变量的值。我们称局部影响变量的集合为马尔可夫毯(Markov blanket)。这种局部性正是信念网络中使用的。信念网络(belief network)是随机变量集合条件依赖的一个直接的模型。在信念网络中条件独立的精确表示要考虑到方向性。

要定义信念网络,首先需要有一个代表模型所有特性的随机变量集。假设这些变量为 $\{X_1, \dots, X_n\}$ 。然后,选择一个变量全序 X_1, \dots, X_n 。

链式法则(命题 6.3)显示了如何将一个合取分解为条件概率:

$$P(X_1 = v_1 \wedge X_2 = v_2 \wedge \dots \wedge X_n = v_n) = \prod_{i=1}^n P(X_i = v_i \mid X_1 = v_1 \wedge \dots \wedge X_{i-1} = v_{i-1})$$

或者,就随机变量和概率分布而言,

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^n P(X_i \mid X_1, \dots, X_{i-1})$$

定义随机变量 X_i 的父节点,记为 $parents(X_i)$,是全序中 X_i 前续节点的最小集合,使得其余 X_i 的前续节点条件独立于给定 $parents(X_i)$ 的 X_i 。即 $parents(X_i) \subseteq \{X_1, \dots, X_{i-1}\}$, 使得

$$P(X_i \mid X_{i-1}, \dots, X_1) = P(X_i \mid parents(X_i))$$

如果存在一个以上的最小集合,任何最小集合可以被选择为父节点。只有当前续节点中的一些是其他节点的确定函数时,才可能存在一个以上的最小集合。

我们可以把链式法则和父节点的定义合在一起,给出

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^n P(X_i \mid parents(X_i))$$

235

所有变量的概率 $P(X_1, X_2, \dots, X_n)$, 叫做联合概率分布(joint probability distribution)。信念网络定义了一个联合概率分布的因式分解(factorization), 其中条件概率组成

其相乘的因子。

信念网络也称为**贝叶斯网络**(Bayesian network), 是一个无向图(DAG), 其中的节点是随机变量。从每个 $parents(X_i)$ 到 X_i 元素有一条弧。与信念网络相关联的是条件概率分布集合——给出其父节点的每个变量的条件概率(其中包括那些无父母变量的先验概率)。

因此, 信念网络的组成为:

- DAG, 其中每个节点被标记为一个随机变量;
- 每个随机变量对应一个域;
- 条件概率分布集合对于每个变量 X 给出 $P(X | parents(X))$ 。

信念网络是无环结构的, 链式法则分解合取给出顺序, 变量只能把前续节点看成是父节点, 不同的分解会导致不同的信念网络。

【例 6-10】 假设我们要使用诊断助手来诊断建筑物中是否发生火灾, 这要基于嘈杂的传感器信息和可能的现场争论的描述。Agent 接收到有关是否每个人都离开大楼的报告。假设报告传感器是嘈杂的: 有时没有大批人离开时它会报告有离开(错误的积极), 有时当所有人都离开了它也不报告(错误的消极)。假设火警响起可以导致离开, 但这不是一个决定性关系。篡改情况或发生火灾都会影响报警。火灾也会引起浓烟从建筑物中升起。

在以下命令中, 假设我们使用的变量全都是布尔型:

- 篡改随着报警发生时, *Tampering* 为真。
- 当发生火灾时, *Fire* 为真。
- 警铃响时, *Alarm* 为真。
- 当有烟时, *Smoke* 为真。
- 如果一旦有大量人口离开建筑物时, *Leaving* 为真。
- 如果有人离开而产生报告, *Report* 为真。如果无人离开而产生报告, 则 *Report* 为假。

变量 *Report* 表示传感器报告有人离开, 这种信息是不可靠的, 因为有人发出这样的报告可能是在玩恶作剧, 或者是给出这样的报告也没有人注意到。引入这个变量允许不可靠的传感器数据。Agent 知道传感器报告了什么, 但关于人们离开大楼它也只是得到不可靠的证据。

作为域的一部分, 假设如下的条件独立:

- *Fire* 是有条件独立于 *Tampering* (没有其他信息的条件下)。
- *Alarm* 取决于 *Fire* 和 *Tampering*。也就是说, 我们不做出有关 *Alarm* 怎样取决于其给出变量顺序的前续节点的假设。
- *Smoke* 只取决于 *Fire*, 它条件独立于 *Tampering* 和 *Fire* 是否发生的 *Alarm*。
- *Leaving* 只取决于 *Alarm*, 与 *Fire* 或 *Tampering* 或 *Smoke* 没有直接关系。也就是说, *Leaving* 条件独立于给出 *Alarm* 的其他变量。
- *Report* 仅直接取决于 *Leaving*。

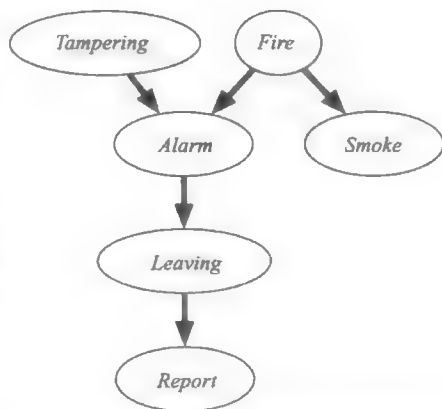


图 6-1 的信念网络表现了这些依赖关系。该网呈现的因式为:

图 6-1 例 6-10 关于离开报告的信念网络

$$\begin{aligned}
 &P(\text{Tampering}, \text{Fire}, \text{Alarm}, \text{Smoke}, \text{Leaving}, \text{Report}) \\
 &= P(\text{Tampering}) \times P(\text{Fire}) \times P(\text{Alarm} | \text{Tampering}, \text{Fire}) \\
 &\quad \times P(\text{Smoke} | \text{Fire}) \times P(\text{Leaving} | \text{Alarm}) \times P(\text{Report} | \text{Leaving})
 \end{aligned}$$

237

我们必须定义每个变量的域。假设变量是布尔型的，也就是说，它们的域为 $\{\text{true}, \text{false}\}$ 。我们用小写形式的变量代表真，用其否定表示假。因此，举例 *Tampering* 为真记为 *tampering*，*Tampering* 为假记为 $\neg \text{tampering}$ 。

以下的例子遵循条件概率假设：

$$\begin{aligned}
 P(\text{tampering}) &= 0.02 \\
 P(\text{fire}) &= 0.01 \\
 P(\text{alarm} | \text{fire} \wedge \text{tampering}) &= 0.5 \\
 P(\text{alarm} | \text{fire} \wedge \neg \text{tampering}) &= 0.99 \\
 P(\text{alarm} | \neg \text{fire} \wedge \text{tampering}) &= 0.85 \\
 P(\text{alarm} | \neg \text{fire} \wedge \neg \text{tampering}) &= 0.0001 \\
 P(\text{smoke} | \text{fire}) &= 0.9 \\
 P(\text{smoke} | \neg \text{fire}) &= 0.01 \\
 P(\text{leaving} | \text{alarm}) &= 0.88 \\
 P(\text{leaving} | \neg \text{alarm}) &= 0.001 \\
 P(\text{report} | \text{leaving}) &= 0.75 \\
 P(\text{report} | \neg \text{leaving}) &= 0.01
 \end{aligned}$$

【例 6-11】 考虑图 1-8 接线的例子。假设我们取灯是否亮的变量，开关位置的变量，灯和开关是否有故障的变量，是否有电源的变量。变量在图 6-2 中定义。

我们选择一个顺序，将导致变量的原因放在变量之前。例如，决定灯是否亮的变量排在灯是否工作正常的变量和灯是否有电的变量之后。

灯 l_1 是否亮只取决于电源线 ω_0 中是否有电，灯 l_1 是否正常工作。其他变量，如开关 s_1 的位置和灯 l_2 是否亮，或谁是加拿大女王，都是不相关的。因此， L_1_lit 的父节点是 ω_0 和 L_1_st 。

考虑变量 W_0 ，表示电源线 ω_0 中是否有电。如果我们知道电源线 ω_1 和 ω_2 中是否有电，知道开关 s_2 的位置和开关是否正常工作，那么其他变量值（除了 L_1_lit ）就不会影响对电源线 ω_0 中是否有电的信念。因此， W_0 的父节点为 S_2_Pos 、 S_2_st 、 W_1 和 W_2 。

图 6-2 显示了考虑每个变量独立性导致的信念网络结果。信念网络还包含变量的域，如图中所示，也包括给出其父节点的每个变量的条件概率。

238

具体说明如下：

- 对于每根电线 ω_i ，有一个随机变量 W_i ，其域为 $\{\text{live}, \text{dead}\}$ ，它表示线 ω_i 中是否有电。 $W_i = \text{live}$ 意味着线 ω_i 中有电。 $W_i = \text{dead}$ 意味着 ω_i 中没电。
- *Outside_power*，其域为 $\{\text{live}, \text{dead}\}$ ，表示建筑物是否来电。
- 对于每个开关 s_i ，变量 S_i_pos 意味着 s_i 的位置，它的域为 $\{\text{up}, \text{down}\}$ 。
- 对于每个开关 s_i ，变量 S_i_st 表示开关 s_i 的状态。它的域为 $\{\text{ok}, \text{upside_down}, \text{short}, \text{intermittent}, \text{broken}\}$ ， $S_i_st = \text{ok}$ 时表示开关 s_i 正常工作。 $S_i_st = \text{upside_down}$ 表示开关 s_i 安装上下颠倒了。 $S_i_st = \text{broken}$ 表示开关 s_i 断开，不允许电流流入。
- 对于每个断路器 cb_i 来说，变量 Cb_i_st 的域为 $\{\text{on}, \text{off}\}$ 。 $Cb_i_st = \text{on}$ 表示 cb_i 可以流入电流， $Cb_i_st = \text{off}$ 表示 cb_i 不可以流入电流。
- 对于每个灯 l_i ，变量 L_i_st 的域为 $\{\text{ok}, \text{intermittent}, \text{broken}\}$ ，表示灯的状态。

$L_i_st=ok$ 表示灯 l_i 如果有电就会亮, $L_i_st=intermittent$ 表示灯 l_i 如果有电会间歇性亮起, $L_i_st=broken$ 表示灯 l_i 不能正常工作。

对于变量 W_1 , 规定了下面的条件概率:

$$P(W_1=live | S_1_pos=up \wedge S_1_st=ok \wedge W_3=live)$$

$$P(W_1=live | S_1_pos=up \wedge S_1_st=ok \wedge W_3=dead)$$

$$P(W_1=live | S_1_pos=up \wedge S_1_st=upside_down \wedge W_3=live)$$

⋮

$$P(W_1=live | S_1_pos=down \wedge S_1_st=broken \wedge W_3=dead)$$

S_1_pos 有 2 个值, S_1_ok 有 5 个值, W_3 有 2 个值, 所以指定一个值给 $W_1=live$ 时有 $2 \times 5 \times 2 = 20$ 种不同的情况。只要考虑到概率论, 这 20 种情况下 $W_1=live$ 的概率可以任意分配。当然, 域知识限制了什么值才是有意义的。 $W_1=dead$ 的值可以从每一个这些情况中的 $W_1=live$ 的值中计算得出。

由于变量 S_1_st 没有父节点, 它需要指定所有概率的一个先验分布, 除了其中的一个值。剩余值可以由所有概率和为 1 的约束来推导。因此, 要指定 S_1_st 的分布, 必须指定以下 5 个概率中的 4 个:

$$P(S_1_st = ok)$$

$$P(S_1_st = upside_down)$$

$$P(S_1_st = short)$$

$$P(S_1_st = intermittent)$$

$$P(S_1_st = broken)$$

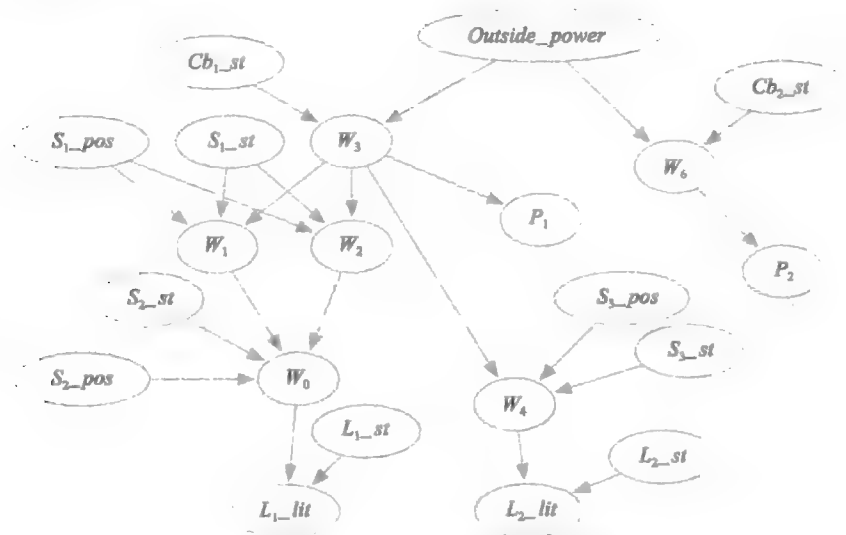


图 6-2 图 1-8 中电气领域的信念网络

其他变量类似表示。

信念网络是一个条件独立的图形表示。其独立性允许我们描述出其对图形的直接影响, 并规定哪些概率必须指定。任意的后验概率可以从网中推得。

嵌入在信念网络中的独立性假设如下: 每个随机变量条件独立于给出其父节点的子节点。也就是说, 如果随机变量 X 的父节点为 Y_1, \dots, Y_n , 那么, 所有不是 X 的子节点的随机变量都条件独立于给定 Y_1, \dots, Y_n 的 X :

$$P(X | Y_1, \dots, Y_n, R) = P(X | Y_1, \dots, Y_n)$$

如果 R 不包含 X 的子节点。对于这个定义,我们将 X 包含于其子节点本身。等式右侧是信念网络部分的概率的形式。只要不是 X 的子节点, R 可能包含 X 的前面的节点和其他节点。独立性假设指出,知道了 X 的父节点就可以获取所有非子节点变量的影响。

240

通常情况下,我们就将 DAG 作为一个信念网络。这样做,记住每个变量的域和条件概率分布集很重要,也是网络中的一部分。

对于每个变量指定的概率数在变量父节点数目中是指数的。独立性假设是有用的,因为直接影响另一个变量的变量数是很小的。应该将变量进行排序,这样节点就可以拥有尽可能少的父节点。

信念网络和因果关系

信念网络经常被称为**因果网络**(causal network),并且是因果关系的一种很好的表示。因果模型可以预测中间结果。假设你想到一个域的因果模型,此域是就随机变量集合而指定的。对于每对随机变量 X_1 和 X_2 ,如果从 X_1 到 X_2 存在直接的因果关系,从 X_1 到 X_2 添加一个弧(例如,在其他变量语境中改变 X_1 可以间接影响到 X_2 ,这不能通过获取这些中间变量来建立模型)。我们希望这个因果模型能够遵从信念网络中的独立性假设。因此,所有信念网络的结论是有效的。

我们也期望这样的图是无环的。如果考虑到随机变量是代表特定事件,而非事件的类型,那么这种假设是合理的。例如,考虑一个因果链,“有压力”导致“工作效率低下”,反过来,导致“压力更大”。为了打破这种明显的周期性循环,我们可以在不同的阶段表现“有压力”,就好像不同的时间有不同的随机变量一样。过去某时有压力会使你不能正常工作,也导致在未来也会有压力。变量应满足**明确性原则**(clarity principle)和一个明确的意义。变量不应该被看做是事件类型。

信念网络本身没有什么因果关系,它可以代表非因果独立性,但它似乎特别适合于域中存在因果关系的情况。添加代表局部的因果关系的弧,可以产生一个小信念网络。图 6-2 的信念网络表示出对于一个简单域来说这是怎么形成的。

因果网络模拟了**干预因素**(intervention)。如果人为地给一个变量赋一个特定的值,变量的子节点(不是其他节点)会受到影响。

最后,可以看到信念网络中的因果关系与 5.7 节中讨论的因果关系和证据推理有怎样的关系。因果信念网络可以看做在因果方向上的一种公理化方式。信念网络中的推理对应于推出原因并从中作出预测。5.7 节中讨论的基于逻辑的外展观点和信念网络两者之间存在一种直接映射:信念网络可以在可能假设下以某种概率模拟逻辑程序。这将在 14.3 节描述。

241

注意在信念网络独立性定义的约束“每个随机变量条件独立于给出其父节点的非子节点”。如果 R 包含变量 X 的子节点,独立性假设是不能直接应用的。

【例 6-12】 在图 6-2 中,变量 S_3_pos 、 S_3_st 和 W_3 是变量 W_1 的父节点。如果知道 S_3_pos 、 S_3_st 和 W_3 的值,那么 l_1 是否亮, Cb_1_st 的值是多少都不会影响对于电源线 ω_1 中是否有电的信念。但是,即使知道了 S_3_pos 、 S_3_st 和 W_3 的值,知道 l_2 是否亮会潜在地改变电源线 ω_1 是否有电的信念,独立性假设不能直接应用。

变量 S_1_pos 没有父节点,因此,信念网络中的嵌入的独立性指定,对于任意给定的 A , $P(S_1_pos=up|A)=P(S_1_pos=up)$, A 不涉及 S_1_pos 的子节点。例如,如果 A 包括 $S_1_pos=up$ 的子节点,比如说 A 是 $S_2_pos=up \wedge L_1_lit=true$,独立性假设不能直接应用。

信念网络规定从任意条件概率可以推得联合概率分布。在网络中可以在任意其他变量值的条件限制下查询任意变量的条件概率。这通常是通过对一些变量进行观察并且查询其他变量来完成。

【例 6-13】考虑例 6-10。每个变量的先验概率(没有证据情况下)可以使用下一节的方法来计算。下面的条件概率遵从例 6-10 的模型,保留小数点后三位:

$$P(\text{tampering}) = 0.02$$

$$P(\text{fire}) = 0.01$$

$$P(\text{report}) = 0.028$$

$$P(\text{smoke}) = 0.0189$$

观察报告给出了以下结果:

$$P(\text{tampering} | \text{report}) = 0.399$$

$$P(\text{fire} | \text{report}) = 0.2305$$

$$P(\text{smoke} | \text{report}) = 0.215$$

正如预期的那样,通过此报告, *tampering* 和 *fire* 的概率都增加。因为 *fire* 的概率增加, *smoke* 的概率也增加了。

假定观察到 *smoke*:

$$P(\text{tampering} | \text{smoke}) = 0.02$$

$$P(\text{fire} | \text{smoke}) = 0.476$$

$$P(\text{report} | \text{smoke}) = 0.320$$

注意,观察到 *smoke* 不会影响到 *tampering* 的概率;然而, *report* 和 *fire* 的可能性会增加。

假设 *report* 和 *smoke* 都观察到了:

$$P(\text{tampering} | \text{report} \wedge \text{smoke}) = 0.0284$$

$$P(\text{fire} | \text{report} \wedge \text{smoke}) = 0.964$$

观察两者更能确定 *fire* 的发生率。然而在 *report* 情况下, *smoke* 的出现会使 *tampering* 的概率更小。这是因为 *report* 是出自对 *fire* 的解释,导致真实情况发生率更高。

假设现在观察到的是 *report* 并非是 *smoke*:

$$P(\text{tampering} | \text{report} \wedge \neg \text{smoke}) = 0.501$$

$$P(\text{fire} | \text{report} \wedge \neg \text{smoke}) = 0.0294$$

在 *report* 情况下, *fire* 的概率小,所以 *tampering* 增加概率来解释 *report*。

这个例子说明了信念网络独立性假设是怎样给出常识性的结论,以及为什么是一种信念网络独立性假设的结果。

这个网络可以以多种方式来使用:

- 给定条件如下:开关和断路器正常,外部电源值和开关位置已知。该网络可以模拟灯是如何工作的。
- 给定外部电源的值和开关的位置,网络可以推断出任何输出的可能性——例如, l_1 亮的概率。
- 给定开关的值和灯是否亮起,每个开关或断路器在任何特定状态的后验概率可以推断出来。
- 给出一些观察,可以使用网络来进行回推从而决定最佳开关位置。
- 给出开关位置、输出和一些中间值,可以使用网络来决定网络中任意其他变量的概率。

构建信念网络

要代表一个信念网络中的域，网络设计师必须考虑以下问题：

1) 相关的变量是什么？设计师要特别考虑：

- 在域中 Agent 会观察到什么，观察到的每一点都是一个随机变量，因为 Agent 必须能对它所观察到的每样东西都能控制。
- 给定观察，Agent 要对它所感兴趣的信息知道其概率，每个特点都要视为变量来获得。
- 其他隐藏的变量(hidden variable)或潜在的变量(latent variable)不会被观察到或查询到，但是这使得模型更简单化。这些变量要么导致依赖关系，要么减少条件概率规格的大小。

243

2) 这些变量应该取什么样的值？这需要考虑 Agent 推理达到的详细程度，来回答所遇到查询的种类。

对于每个变量来说，设计人员应该指定其域中所采用值的含义。在域中对于一个有特殊值的变量来说什么必须为真应该满足明确性原则。明确记录所有变量的含义和它们可能的值是一个好主意。只有在 Agent 想从数据中学习的隐藏变量的值存在时，设计者才不想这么做(见 11.2.2 节)。

3) 变量之间是什么关系？这应由本地影响力来表示，并且使用亲子关系进行建模。

4) 变量的分布是怎样依赖于本地影响它的变量的(其父节点)？这是在条件概率分布下表示的。

【例 6-14】 假设希望诊断助手能够推断出患者的气喘和咳嗽的可能原因，如例 5-30。

- Agent 可以观察到咳嗽、气喘、发烧，并且询问病人是否吸烟。因此，有对应这些因素变量。
- Agent 希望了解其他病人的症状和各种可能的治疗方法；如果是这样，这些也应该是变量(尽管它们在本例中不使用)。
- 存在有用的变量可预测出病人的结果。医学界已命名的这些特点及对其症状进行特征化。在这里，我们将使用支气管炎和流感变量。
- 现在考虑这些变量直接依赖于什么。患者是否哮喘取决于他们是否有支气管炎。他们是否咳嗽取决于他们是否有支气管炎。患者是否有支气管炎取决于他们是否有流感，以及他们是否吸烟。他们是否有发烧取决于他们是否有流感。图 6-3 描述了这些依赖关系。

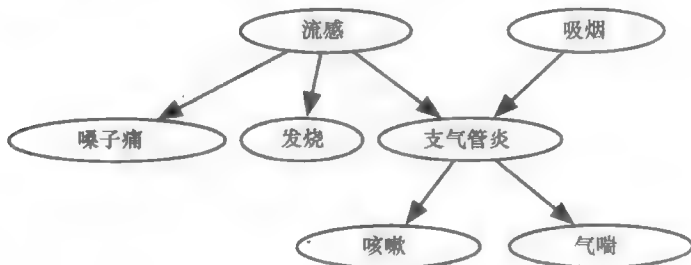


图 6-3 例 6-14 的信念网络

- 选择变量的值，涉及要考虑到所推断东西的详细程度。可以给每个疾病或症状的严重程度进行编码，把它们作为变量的值。例如，对于气喘变量可以使用重度、

244

中度、轻度，或无气喘。例如，通过对疾病的所有图形进行表征，可以在一个较低的抽象层次建立疾病的模型。为了便于说明，我们将在一个非常抽象的层次上建立一个域，只考虑症状和疾病的存在或不存在。每一个变量将其域为 $\{true, false\}$ 的布尔值，表示相关的疾病或症状的存在或不存在。

- 评定每个变量是怎样依赖于其父节点，这是要靠给出其父节点的每个变量的条件概率来给定：

$$P(\text{流感})=0.05$$

$$P(\text{吸烟})=0.2$$

$$P(\text{喉咙痛} \mid \text{流感})=0.3$$

$$P(\text{喉咙痛} \mid \text{非流感})=0.001$$

$$P(\text{发烧} \mid \text{流感})=0.9$$

$$P(\text{发烧} \mid \text{非流感})=0.05$$

$$P(\text{支气管炎} \mid \text{流感} \wedge \text{吸烟})=0.99$$

$$P(\text{支气管炎} \mid \text{流感} \wedge \text{不吸烟})=0.9$$

$$P(\text{支气管炎} \mid \text{非流感} \wedge \text{吸烟})=0.7$$

$$P(\text{支气管炎} \mid \text{非流感} \wedge \text{不吸烟})=0.0001$$

$$P(\text{咳嗽} \mid \text{支气管炎})=0.8$$

$$P(\text{咳嗽} \mid \text{无支气管炎})=0.07$$

$$P(\text{气喘} \mid \text{支气管炎})=0.6$$

$$P(\text{气喘} \mid \text{无支气管炎})=0.001$$

通过对症状观察和故障或疾病的后验概率来进行诊断(diagnosis)过程。

这个例子也说明了另一个喜欢辩解并且与复杂诊断相比更偏爱简单诊断的例子。

在任何观察之前，我们可以计算(见下一节)几个显著的数字， $P(\text{吸烟})=0.2$ ， $P(\text{流感})=0.05$ ， $P(\text{支气管炎})=0.18$ 。一旦观察到气喘，以上三者发生的概率更高： $P(\text{吸烟} \mid \text{气喘})=0.79$ ， $P(\text{流感} \mid \text{气喘})=0.25$ ， $P(\text{支气管炎} \mid \text{气喘})=0.992$

假设观察到气喘 \wedge 发烧： $P(\text{吸烟} \mid \text{气喘} \wedge \text{发烧})=0.32$ ， $P(\text{流感} \mid \text{流感} \wedge \text{发烧})=0.86$ ， $P(\text{支气管炎} \mid \text{气喘} \wedge \text{发烧})=0.998$ 。注意，像例 5-30 那样，当观察到发烧时，是怎样暗示出有流感的，同样也揭示了吸烟的行为。

【例 6-15】 考虑图 6-2 中所描述的信念网络。注意到，这个模型中嵌入的独立性假设：DAG 指出灯、开关、断路器是独立坏掉的。为了建立开关是如何坏掉的依赖性模型，可以添加更多的弧也可以是更多的节点。例如，如果灯没有独立性的坏掉，那是因为它们是同一批生产的，可以添加额外的节点来传达这种依赖关系，可以添加一个节点表示这些灯都来自一批好的生产还是坏的生产，其父节点是 L_1_st 和 L_2_st 。该灯现在可以依赖性地坏掉了。当有证据证明，一个灯坏了，那么它同一批次为坏的可能性会增加，从而其他灯也是坏的可能性就更大了。如果不确定这些灯是否来自同一批次，也可以添加一个节点来表示。重点是信念网络提供了一种独立性的规格，可以让我们以一种自然、直接的方式来建立依赖性模型。

该模型表明，没有电线短路或是房子接线与图不同的可能性。特别是，它表明 ω_0 对于 ω_1 来说不会短路，所以 ω_0 从 ω_1 中来电。可以添加额外的依赖关系，给每一个可能的短期建模。另一种方法是增加一个额外的节点，表明该模型是合适的。从这个节点的弧会导致每个变量代表电线的电源和每个灯中的电源。当模型适合时，可以使用例 6-11 的概率。当模型不适合时，可以指定每根电线和灯随机地工作。当存在与原始模型不适合的观察时，就不可能或肯定不能给出模型，模型不适合的概率会增加。

【例 6-16】 假设开发一个帮助系统(help system)，基于用户在查询帮助系统输入的关键词来确定用户感兴趣的帮助页。

系统将观察用户提供的词。假设我们不想建立句子结构，但是假设词语集就足以确定帮助页面，用户可以给定多个单词。一种表示方法是给每一个词语赋一个布尔变量，因此，当用户使用查询时，标有 able、absent、add、zoom……的词语其值为真，当用户不使用时，其值为假。

我们感兴趣的是用户想要哪种帮助页面。假设用户仅对一个帮助页面感兴趣，因此，构造一个节点 H ，其域为所有帮助页面 $\{h_1, \dots, h_k\}$ 的集合是合理的。

一种表示方法是朴素贝叶斯分类(naive Bayesian classifier)。朴素贝叶斯分类器是一个拥有单节点(类)的信念网络，其直接影响其他变量，并且其他变量对于给出的类是独立的。图 6-4 显示了一个有关帮助系统的朴素贝叶斯分类器，其中用户感兴趣的帮助页面 H 为一个类，其他节点代表用户在查询中使用的词语。在这个网络中，查询中所用的词依赖于用户感兴趣的帮助页面，并且这些词对于给定的帮助页面是相互条件依赖的。

246

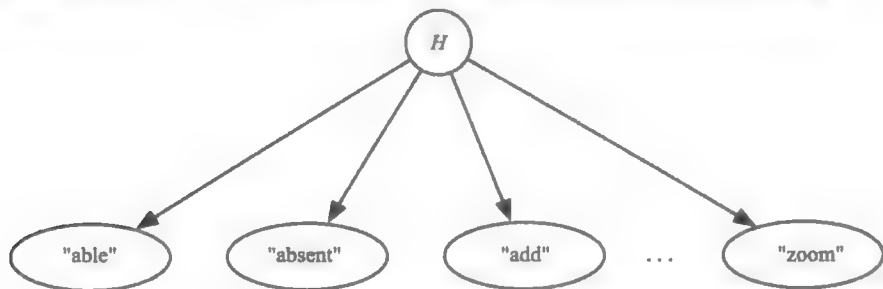


图 6-4 例 6-16 的朴素信念网络

对于每个帮助页面 h_i ，在没有任何信息的情况下，网络用 $P(h_i)$ 表示用户想要帮助页面 h_i 的程度。用户有一些特殊问题需要帮助，根据帮助的程度，可以获得信息。网络假设用户真实地对一个页面感兴趣，因此此时 $\sum P(h_i) = 1$ 。

网络也要求，对于每个词 w_j 和每个帮助页面 h_i ，概率用 $P(w_j | h_i)$ 表示。这些值看起来似乎很难查询，但是我们可以使用几个启发式方法。这些数的平均值可用查询中词的平均数目除以词的总数。当询问帮助页面时，我们希望出现在帮助页面的词比不出现在帮助页面的词更加可用。可能存在和人们经常使用的页面相关联的关键词，也可能存在这样一些词，它们经常被使用，但是独立于用户感兴趣的帮助页面。例 7-13 显示了网络的概率分布怎样从经验中得到。

对于查询中词的集合，出现在查询中的词被视为真，未出现在查询中的词被视为假。例如，假如帮助文本是“the zoom is absent”，词“the”、“zoom”、“is”、“absent”将被赋予真，其他的词将被赋予假。关于 H 的后验分布被计算，且最有可能的几个帮助主题能够显现给用户。

一些词，如“the”和“is”可能不如那些和帮助主题相关，但是和它们有相同条件概率的其他词汇，这些词在模型中会被去除。没有被预期到出现在查询中的词也会从模型中去除。

注意到，条件是词没有出现在查询中。例如，如果 h_{73} 页是关于打印问题，我们可能期望想要 h_{73} 页的用户使用“print”这个词。在查询中“print”词不存在是用户不想要 h_{73} 页面

的一个强有力的证明。

247

帮助页面中词的独立性是一个很强的假设。这个假设并没有应用到像“not”这样的词上,在这种情况下,词“not”和什么相关是重要的。假如人们正在问一些句子,句子中的词不会条件独立于句中的其他词,因为一个词的概率依赖于句中的上下文环境。可能存在一些互相补充的词,在这种情况下,希望用户使用其中的一个而不使用另一个(例如“type”和“write”),和希望一起使用的词(例如,“go”和“to”)。这些情况都违反了独立性假设。至于违反假设什么程度会破坏系统的实用性,这是一个经验的问题。◀

6.4 概率推理

通常,主要的概率推理任务是在给定一些证据条件下,计算查询变量的后验概率分布。不幸的是,即使在绝对误差(不到 0.5)或固定的乘数因子条件下,信念网络中后验概率的评估问题也是一个 NP 难问题,因此,一般的高效执行将不会有效。

信念网络中概率推理的主要方法如下:

- 利用网络的结构。这种方法是变量消除算法的典型代表,以后将详细说明。
- 基于搜索的方法。通过列举一些可能的领域,后验概率可以从生成的领域估测到。通过计算没有考虑到的领域的概率分布,在后验概率中误差的范围可以被估计到。这种方法适用于分布是极端的情况(所有的概率接近于 0 或接近 1),如发生在工程系统中。
- 变分推理(variational inference),它的思想是找到一个容易计算的问题的近似值。首先选择一个比较容易计算的表示类。这个类要和不连通的信念网络的集合一样简单(没有弧)。下一步,试图找到一个最接近原始问题的类成员。也就是说,找到一个容易计算的分布,这个分布要尽可能地接近后验分布且这个分布能够被计算。因此,问题简化为最小化误差的最优化问题。
- 随机模拟。在这些方法中,根据概率分布产生随机案例。通过将这些随机案例当做一系列的样本集合,此时关于任意变量组合的边缘分布能够被估计。在 6.4.2 节我们介绍了随机模拟方法。

在本书中,我们仅仅介绍了第 1 种和第 4 种方法。

6.4.1 信念网络中的变量消除

248

这一部分介绍了在一个任意结构的信念网络中,找到关于一个变量的后验分布的算法。许多准确有效的算法能被视为此算法的优化。这个算法可以视为满足约束问题(CSPs)的变量消除(VE)算法或者满足软约束的 VE 算法的一个变异版本。

此算法建立在一种思想之上,即信念网络中特别指出了联合概率分布的一个因式分解。

在给出此算法之前,我们定义了一些因式和操作,进而以它们为操作对象。 $P(X|Y)$ 是一个包含变量 X 和 Y 的实数变量,对于给定的 X 值和 Y 值,其表示对于给定值 Y ,关于 X 值的条件概率。正如因式的思想,变量函数的思想由此产生了。在信念网络中,VE 算法通过控制因式来计算后验概率。

一个因式表示从一个随机变量元组映射到一个实数的函数。对于 $f(X_1, \dots, X_i)$, 变量 X_1, \dots, X_i 表示因式 f 的变量且 f 是关于 X_1, \dots, X_i 的一个因式。

设想 $f(X_1, \dots, X_i)$ 是一个因式, 每个 v_i 是 X_i 域的一个元素。当每个 X_i 有一个 v_i 值时, $f(X_1=v_1, X_2=v_2, \dots, X_i=v_i)$ 就表示一个数。因式的一些变量能被赋值进而产生一个关于其他变量的新因式。例如, $f(X_1=v_1, X_2, \dots, X_i)$ 有时记为 $f(X_1, X_2, \dots, X_i)_{X_1=v_1}$, 它是关于 X_2, \dots, X_i 的一个因式, v_1 是 X_1 变量域的一个元素。

【例 6-17】图 6-5 以图表形式显示了关于变量 X, Y, Z 的一个因式 $r(X, Y, Z)$, 它假设每个变量拥有一个二值域 $\{t, f\}$ 。图中给出了关于 Y, Z 的因式 $r(X=t, Y, Z)$ 的图表。类似的, $r(X=t, Y, Z=f)$ 是关于 Y 的因式, $r(X=t, Y=f, Z=f)$ 是一个数。

$r(X, Y, Z) =$	X	Y	Z	val
	t	t	t	0.1
	t	t	f	0.9
	t	f	t	0.2
	t	f	f	0.8
	f	t	t	0.4
	f	t	f	0.6
	f	f	t	0.3
	f	f	f	0.7

$r(X=t, Y, Z) =$	Y	Z	val
	t	t	0.1
	t	f	0.9
	f	t	0.2
	f	f	0.8

$r(X=t, Y, Z=f) =$	Z	val
	t	0.9
	f	0.8

$r(X=t, Y=f, Z=f) = 0.8$

图 6-5 因式和赋值的例子

因式能做相乘运算。假设 f_1 和 f_2 是因式, f_1 是包含变量 X_1, \dots, X_i 和 Y_1, \dots, Y_j 的因式, f_2 是包含变量 Y_1, \dots, Y_j 和 Z_1, \dots, Z_k 的因式。 Y_1, \dots, Y_j 是 f_1 和 f_2 的公共变量, f_1 和 f_2 的乘积(product)记为 $f_1 \times f_2$, 它是关于变量单元 $X_1, \dots, X_i, Y_1, \dots, Y_j, Z_1, \dots, Z_k$ 的因式, 定义如下:

$$(f_1 \times f_2)(X_1, \dots, X_i, Y_1, \dots, Y_j, Z_1, \dots, Z_k) = f_1(X_1, \dots, X_i, Y_1, \dots, Y_j) \times f_2(Y_1, \dots, Y_j, Z_1, \dots, Z_k)$$

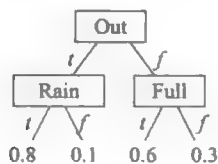
条件概率及因式表示

条件概率被视为是相关变量的函数。一个因式表示关于变量的函数, 因此, 一个因式能用来表示条件概率。

当变量有有限的域时, 这些因式能被作为矩阵执行。如果存在变量的排序(如, 按字母次序), 区域内的值能被映射为非负实数, 存在每个元素的典型表示一维数组, 此时它能够通过自然数被检索。通过使用这样的表示, 关于因式的操作能够有效地被执行。然而, 对用户来说, 这并不是最好的表示方法, 因为条件的结构丢失了。

因式不必作为数组执行。当有太多的父节点时, 用列表表示太大。通常, 大多数结构存在能够被使用的条件概率。

一个这样的结构使用上下文特定的独立(context-specific independence), 在这个环境中, 已知第三个变量的特殊值, 此时一个变量条件独立于另外一个变量。例如, 设想机器人能够走到外面或者获得咖啡。它是否会淋湿取决于环境中是否有雨, 在这里, 环境指的是它出去或者如果它得到咖啡后杯子是否是满的。存在许多表示条件概率 $P(Wet | Out, Rain, Full)$ 的方法, 如用决策树表示, 用概率规则表示, 用带有环境的图表表示。



$wet \leftarrow out \wedge rain: 0.8$
 $wet \leftarrow out \wedge \sim rain: 0.1$
 $wet \leftarrow \sim out \wedge full: 0.6$
 $wet \leftarrow \sim out \wedge \sim full: 0.3$

out:

Rain	Wet	Prob
t	t	0.8
t	f	0.2
f	t	0.1
f	f	0.9

~ out:

Full	Wet	Prob
t	t	0.6
t	f	0.4
f	t	0.3
f	f	0.7

另一种常见的表示是噪声。例如，设想机器人由于下雨、咖啡、孩子而淋湿。如果天气有雨，那么存在一个由于下雨机器人淋湿的概率。如果机器人有咖啡，那么存在一个由于咖啡机器人淋湿的概率，依此类推。如果由于其中一个原因机器人淋湿，那么就用 or 连接。

下一章探讨了表示概率分布的其他方法。

【例 6-18】 图 6-6 显示了 $f_1(A, B)$ 和 $f_2(B, C)$ 的乘积，它是关于 A, B, C 的一个因式。注意， $(f_1 \times f_2)(A=t, B=f, C=f) = f_1(A=t, B=f) \times f_2(B=f, C=f) = 0.9 \times 0.4 = 0.36$ 。

$f_1 =$	A	B	val
	t	t	0.1
	t	f	0.9
	f	t	0.2
	f	f	0.8

$f_2 =$	B	C	val
	t	t	0.3
	t	f	0.7
	f	t	0.6
	f	f	0.4

 $f_1 \times f_2 =$

A	B	C	val
t	t	t	0.03
t	t	f	0.07
t	f	t	0.54
t	f	f	0.36
f	t	t	0.06
f	t	f	0.14
f	f	t	0.48
f	f	f	0.32

图 6-6 因式相乘的例子

余下的操作是计算出因式中的一个变量。针对因式 $f(X_1, \dots, X_j)$ ，统计出一个变量，记为 X_1 ，关于一个因式中的其他变量 X_2, \dots, X_j 的定义如下：

$$\left(\sum_{X_1} f \right) (X_2, \dots, X_j) = f(X_1 = v_1, X_2, \dots, X_j) + \dots + f(X_1 = v_k, X_2, \dots, X_j)$$

其中 $\{v_1, \dots, v_k\}$ 是关于变量 X_1 的可能取值的集合。

【例 6-19】 图 6-7 给出了一个从 $f_3(A, B, C)$ 中计算变量 B 的例子，其中 $f_3(A, B, C)$ 是关于 A 和 C 的因式：

$$\left(\sum_B f_3 \right) (A=t, C=f) = f_3(A=t, B=t, C=f) + f_3(A=t, B=f, C=f) = 0.07 + 0.36 = 0.43$$

条件概率分布 $P(X | Y_1, \dots, Y_j)$ 能够被视为是关于变量 X, Y_1, \dots, Y_j 的一个函数，其中

$$f(X=u, Y_1=v_1, \dots, Y_j=v_j) = P(X=u | Y_1=v_1 \wedge \dots \wedge Y_j=v_j)$$

通常,人们喜欢用 $P(.|.)$ 表示,但是实质上计算机仅仅将条件概率当做因式。

251

$f_3 =$	A	B	C	val
	t	t	t	0.03
	t	t	f	0.07
	t	f	t	0.54
	t	f	f	0.36
	f	t	t	0.06
	f	t	f	0.14
	f	f	t	0.48
	f	f	f	0.32

$\sum_B f_3 =$		A	C	val
		t	t	0.57
		t	f	0.43
		f	t	0.54
		f	f	0.46

图 6-7 从一个因式中和算出一个变量

信念网络推理问题(belief network inference problem)是在给出一些证据的前提下,计算变量后验分布的问题。

计算后验分布的问题能够简化为计算联合概率的问题。已知证据 $Y_1 = v_1, \dots, Y_j = v_j$, 查询变量 Z :

$$P(Z|Y_1 = v_1, \dots, Y_j = v_j) = \frac{P(Z, Y_1 = v_1, \dots, Y_j = v_j)}{P(Y_1 = v_1, \dots, Y_j = v_j)} = \frac{P(Z, Y_1 = v_1, \dots, Y_j = v_j)}{\sum_Z P(Z, Y_1 = v_1, \dots, Y_j = v_j)}$$

因此, Agent 能够计算因式 $P(Z, Y_1 = v_1, \dots, Y_j = v_j)$ 且标准化。注意到这是一个仅仅关于 Z 的因式; 对于给定一个值 Z , 它返回一个数值, 其表示证据和 Z 值的联合概率分布。

设想信念网络的变量是 X_1, \dots, X_n , 为了计算因式 $P(Z, Y_1 = v_1, \dots, Y_j = v_j)$, 从联合分布中计算出其他变量。设想 Z_1, \dots, Z_k 是信念网络中关于其他变量的举例, 也就是

$$\{Z_1, \dots, Z_k\} = \{X_1, \dots, X_n\} - \{Z\} - \{Y_1, \dots, Y_j\}$$

证据和 Z 的联合概率是

$$p(Z, Y_1 = v_1, \dots, Y_j = v_j) = \sum_{Z_k} \dots \sum_{Z_1} P(X_1, \dots, X_n)_{Y_1=v_1, \dots, Y_j=v_j}$$

变量 Z_i 被计算出来, 关于它的排序是一个**消除排序**(elimination ordering)。

要注意这和可能的语义分布是怎样相关联的。存在一个可能领域, 将每个值分配给每个变量。**联合概率分布**(joint probability distribution) $P(X_1, \dots, X_n)$ 给出了每个可能领域的概率。针对 Y_i , VE 算法选择带有观察值的领域且计算出拥有同样 Z 值的可能领域。这和条件概率的定义相关。然而, VE 算法做这些要比计算所有的领域更有效。

252

通过联合概率的规则和信念网络的定义,

$$P(X_1, \dots, X_n) = P(X_1 | \text{parent}(X_1)) \times \dots \times P(X_n | \text{parent}(X_n))$$

这里, $\text{parent}(X_i)$ 是变量 X_i 的父节点的集合。

现在我们将信念网络推理问题简化为由因式的乘积和算出一个变量集合的问题。为了有效地解决这个问题, 我们使用中学学习过的分布规则: 为了计算如 $xy + xz$ 形式的乘积和, 提取出常因子 x , 所求变为 $x(y+z)$ 。这就是 VE 算法的本质。因为代数学中术语的使用, 我们将这些能够做乘积因子的元素都叫做因式。起初时, 因式表示条件概率分布, 但是临时的因式仅仅是关于变量的函数, 这些函数通过加、乘因式得到。

为了计算对于给定观察值的查询变量的后验分布:

1) 对于每个条件概率分布组建一个因式。

2) 消除每个非查询变量:

- 如果变量能被观察到, 在变量出现的每个因式中, 其值被设置为观察值。
- 否则, 变量被计算出来。

3) 将其余的因式相乘, 并标准化。

为了从因式 f_1, \dots, f_k 的乘积和算出变量 Z , 首先将变量分成不包含 Z 的因式 f_1, \dots, f_i 和包含 Z 的因式 f_{i+1}, \dots, f_k 两部分, 然后为加和分配公因式。

$$\sum_Z f_1 \times \dots \times f_k = f_1 \times \dots \times f_i \times \left(\sum_Z f_{i+1} \times \dots \times f_k \right)$$

VE 算法明确地建立了最右因式(就多维数组, 树, 规则集合而言)的表示。

图 6-8 给出了 VE 算法的伪码。消除排序能够被先验给出或者能够在空间中被计算。首先在消除顺序过程中选择观察变量是很有价值的, 因为消除这些变量能够简化问题。

在这里假设查询变量没有被观察到。如果一个特殊值能够被观察到, 对观察值来说, 它的后验概率是 1, 反之为 0。

253

【例 6-20】 参照例 6-10 的查询 $P(\text{Tampering} \mid \text{Smoke} = \text{true} \wedge \text{Report} = \text{true})$ 。消除观察变量 Smoke 和 Report , 其余的因式保留:

条件概率	因式
$P(\text{Tampering})$	$f_0(\text{Tampering})$
$P(\text{Fire})$	$f_1(\text{Fire})$
$P(\text{Alarm} \mid \text{Tampering}, \text{Fire})$	$f_2(\text{Tampering}, \text{Fire}, \text{Alarm})$
$P(\text{Smoke} = \text{yes} \mid \text{Fire})$	$f_3(\text{Fire})$
$P(\text{Leaving} \mid \text{Alarm})$	$f_4(\text{Alarm}, \text{Leaving})$
$P(\text{Report} = \text{yes} \mid \text{Leaving})$	$f_5(\text{Leaving})$

算法忽略了读到的条件概率并且仅仅对因式起作用。临时因式并不总是表示一个条件概率。

设想 Fire 在消除排序中被选择。为了消除 Fire , 搜集包含 Fire 的因式— $f_1(\text{Fire})$ 、 $f_2(\text{Tampering}, \text{Fire}, \text{Alarm})$ 、 $f_3(\text{Fire})$, 将它们相乘, 并且从计算结果和算出 Fire 。

调用 $f_0(\text{Tampering}, \text{Alarm})$ 。在这一阶段, 保留下面的因式:

$f_0(\text{Tampering})$

$f_4(\text{Alarm}, \text{Leaving})$

$f_5(\text{Leaving})$

$f_6(\text{Tampering}, \text{Alarm})$

设想下一步要消除 Alarm 。VE 将包含 Alarm 的因式相乘, 并且从乘积中和算出 Alarm , 将它称为 f_7 :

$$f_7(\text{Tampering}, \text{Leaving}) = \sum_{\text{Alarm}} f_4(\text{Alarm}, \text{Leaving}) \times f_6(\text{Tampering}, \text{Alarm})$$

```

1: procedure VE_BN( $V_S, P_S, O, Q$ )
2:   Inputs
3:      $V_S$ : 变量集合
4:      $P_S$ : 代有条件概率的因子集合
5:      $Q$ : 某些变量上观察值的集合
6:      $Q$ : 查询变量
7:   Output
8:      $Q$  上的后验分布
9:   Local
10:     $F_S$ : 因子集合
11:     $F_S \leftarrow P_S$ 
12:    for each  $X \in V_S - \{Q\}$  使用消除排序 do
13:      if  $X$  被观察到 then
14:        for each  $F \in F_S$  ( $F_S$  包含  $X$ ) do
15:          set  $F$  中  $X$  的观察值到  $O$ 
16:          投影  $F$  到其余变量
17:        else
18:           $R_S = \{F \in F_S; F \text{ 包含 } X\}$ 
19:          令  $T$  为  $R_S$  中因子的乘积
20:           $N_X = \sum_X T$ 
21:           $F_S = F_S \setminus R_S \cup \{N\}$ 
22:        令  $T$  为  $R_S$  中因子的乘积
23:         $N_X = \sum_O T$ 
24:      return  $T/N$ 

```

图 6-8 信念网络的 VE 算法

然后有下面的因式：

$$f_0(\text{Tampering})$$

$$f_5(\text{Leaving})$$

$$f_7(\text{Tampering}, \text{Leaving})$$

消除因式中的 *Leaving* 结果：

$$f_6(\text{Tampering}) = \sum_{\text{Leaving}} f_5(\text{Leaving}) \times f_7(\text{Tampering}, \text{Leaving})$$

为了决定关于 *Tampering* 的分布，将其余的因式相乘：

$$f_9(\text{Tampering}) = f_0(\text{Tampering}) \times f_6(\text{Tampering})$$

关于 *Tampering* 的后验分布用下式表示：

$$\frac{f_9(\text{Tampering})}{\sum_{\text{Tampering}} f_9(\text{Tampering})}$$

注意到式中分母是证据的先验概率。

【例 6-21】 考虑到和前面例子中相同的网络环境，对于下面的查询：

$$P(\text{Alarm} | \text{Fire} = \text{true})$$

当 *Fire* 被消除时，因式 $P(\text{Fire})$ 变成了一个不含变量的因式，它仅仅是一个值 $P(\text{Fire} = \text{true})$ 。

设想下一步要消除 *Report*。就是在一个因式中，因式用 $P(\text{Report} | \text{Learning})$ 表示。和算出所有的 *Report* 值，这样就得到了关于 *Leaving* 的一个因式，它的所有值都为 1。这是因为对于任意的 *Leaving* 的值 v ， $P(\text{Report} = \text{true} | \text{Leaving} = v) + P(\text{Report} = \text{false} | \text{Leaving} = v) = 1$ 。

类似的，如果下一步要消除 *Leaving*，可以将变量值为 1 的因式和因式 $P(\text{Leaving} | \text{Alarm})$ 相乘，然后计算出 *Leaving*。此操作再一次产生了所有元素为 1 的因式。

类似的，消除 *Smoke* 会产生不含变量的因式，它的值为 1。注意到即使 *Smoke* 已经被观察到，消除 *Smoke* 将导致产生不含变量的因式，这一点并不会影响关于 *Alarm* 的后验分布。

最终，只存在表示其先验概率的有关 *Alarm* 的因式，并且在标准化过程中常因式将被消除。

算法的复杂度取决于网络复杂性的度量。用表格表示的因式的大小和因式中变量的数目呈指数。对于给定的消除顺序，网络的树宽 (treewidth) 是因式中变量的最大数目，这个因式是在特定的消除顺序下通过统计变量得到的。对所有的消除顺序而言，信念网络的树宽是树宽的最小值。树宽取决于图的结构并且它是图稠密度的一个度量。VE 算法的复杂度在树宽上呈指数增长，在变量数目上呈线性增长。找到最小树宽的消除顺序是一个 NP 难问题，但是正如对 CSP VE 的讨论，存在一个好的消除顺序启发式算法。

存在两种主要的方法能够提高算法的速度。其一是对于给定的观测和查询，可删除不相关变量。另外，可以将图形编译成二级结构，该结构允许高速缓存的图形值。

6.4.2 通过随机模拟进行近似推理

许多问题由于太复杂而不能得到准确的推理，并且我们必须诉诸于近似推理。一个最有效的方法是从相关的网络的后验分布中得到随机样例。

随机模拟 (stochastic simulation) 的基本思想是使用样例集合计算概率。例如， $P(a) = 0.14$ 意味着，1 000 个样例中，约 140 个中的 a 值为真。你可以从样例中得到概率，也可

以根据概率分析样例。

我们考虑 3 个问题：

- 怎样产生样本。
- 怎样纳入观察值。
- 怎样根据样本推断概率。

我们检验了 3 种，使用样本计算变量后验分布的方法：1) 舍选采样；2) 重要性采样；3) 粒子滤波。

256

1. 单个变量的采样

为了从单个离散或者实数变量 X 中产生样本，首先将 X 域中的值排序。对于离散变量，如果没有常规的排序要求，你就可以建立一个任意的序列。考虑到排序，累积概率分布(cumulative probability distribution)是关于 x 的函数，定义为 $f(x) = P(X \leq x)$ 。

为了产生 X 的随机样本，从区间 $[0, 1]$ 中随机选择一个数 y 。为了确保区间 $[0, 1]$ 的每个数有相同被选择的机会，我们从均匀分布中选择 y 。 v 表示 X 的值，累积概率分布中它以某种规则映射到 y 。也就是说， v 是 $\text{dom}(X)$ 的元素即 $f(v) = y$ ，或者说 $v = f^{-1}(y)$ 。然后 $X = v$ 是 X 的一个随机样本，它依据 X 的分布被选择。

【例 6-22】 随机变量 X 的域为 $\{v_1, v_2, v_3, v_4\}$ 。设想 $P(X=v_1)=0.3$, $P(X=v_2)=0.4$, $P(X=v_3)=0.1$, $P(X=v_4)=0.2$ 。首先，将值进行排序， $v_1 < v_2 < v_3 < v_4$ 。图 6-9 显示了 X 的分布 $P(X)$ 和 X 的累积概率分布 $f(X)$ 。将 f 值逆映射到 v_1 的概率是 0.3。因此，如果样本是从 Y 轴均匀选择的， v_1 有 0.3 的概率被选择， v_2 有 0.4 的概率被选择，依此类推。

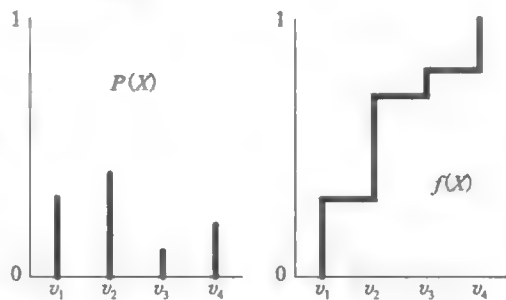


图 6-9 累积概率分布

2. 信念网络中的前向采样

前向采样是信念网络中产生每个变量样本的一种方法，因此每个样本依其概率成比例地产生。假设 X_1, \dots, X_n 是变量的完整排序，因此在变量出现在完整排序中之前，变量的父节点已经出现。前向采样通过某种顺序得到每个变量的 X_1, \dots, X_n 的样本，进而得到所有变量的样本。首先，使用上述算法建立 X_1 的样本。对于其他的每个变量，由于变量的整个排序，当建立 X_i 的样本之前， X_i 的父节点的样本已经建立。已知 X_i 的父节点已被赋值时，从 X_i 的分布中为 X_i 建立样本。对每个变量重复上述步骤，进而产生包含所有变量的样本值。分配概率是选择特殊值分配给所有变量的概率。

257

【例 6-23】 图 6-1 建立了信念网络的样本集合。假设变量排序如下：Tampering, Fire, Alarm, Smoke, Leaving, Report。首先算法使用累积分布的逆建立 Tampering 的样本。假设选择 Tampering=False。然后通过相同的方法为 Fire 建立样本。假设选择 Fire=true，然后使用分布 $P(\text{Alarm} | \text{Tampering} = \text{false}, \text{Fire} = \text{true})$ 对 Alarm 选择一个值。假设选择 Alarm=true。接下来通过使用 $P(\text{Smoke} | \text{Fire} = \text{true})$ 为 Smoke 选择一个值。然后通过使用分布 $P(\text{Leaving} | \text{Alarm} = \text{true})$ 为 Leaving 选择一个值。假设选择 Leaving=false。然后通过使用分布 $P(\text{Report} | \text{Leaving} = \text{false})$ 为 Report 选择一个值。因此为每个变量选择一个值并且创建图 6-10 的第一个样本。注意到它选择了不可能值的一个组合。这并不经常发生，它的发生和样本程度成比例。重复直到有足够的样本。

图 6-10 中产生了 1000 个样本。

样本	Tampering	Fire	Alarm	Smoke	Leaving	Report
s ₁	false	true	true	true	false	false
s ₂	false	false	false	false	false	false
s ₃	false	true	true	true	true	true
s ₄	false	false	false	false	false	true
s ₅	false	false	false	false	false	false
s ₆	false	false	false	false	false	false
s ₇	true	false	false	true	true	true
s ₈	true	false	false	false	false	true
...						
s ₁₀₀₀	true	false	true	true	false	false

图 6-10 信念网络的采样

3. 从采样到概率

从样例集合中使用样本均值估测概率。命题 α 的样本均值 (sample average) 是包含 α 为真的样本的数目除以总的样本数目。随着在大数量样本的分布中样本数目接近无穷值, 样本的均值也就越接近真实的概率。

258

已知 n 个数目的样本, Hoeffding 不等式 (Hoeffding's inequality) 提供了非条件概率误差的一个估测值。

命题 6.6 (Hoeffding) 设想 p 是真值, s 是 n 个独立样本的样本均值, 那么

$$P(|s - p| > \epsilon) \leq 2e^{-2n\epsilon^2}$$

此引理显示了拥有多少样本就可以近似得到概率的一个准确的估测值。为了保证误差 $\epsilon < 0.5$, 我们需要更多的样本。然而, 在 δ 部分样本中, 如果允许大于 ϵ 的误差出现, 可以使用 $2e^{-2n\epsilon^2} < \delta$, 其中 n 满足下面不等式,

$$n > \frac{-\ln \frac{\delta}{2}}{2\epsilon^2}$$

例如, 想要使误差小于 0.1, 也就是说, 在样本的 5% 中, 仅仅愿意容忍误差比 0.1 大一点。可以使用 Hoeffding 的约束, 设置参数 $\epsilon = 0.1$, $\delta = 0.05$, 其中 $n > 184$ 。因此, 可以保证在 185 个样本中, 对出现的误差做上面的约束。如果想要使至少 95% 的样本中, 误差小于 0.01, 需要 18445 个样本。若想要使 99% 的样本中, 误差小于 0.1, 需要 265 个样本。

4. 舍选采样

对于证据 e , 舍选采样使用下式估测 $P(h|e)$:

$$P(h|e) = \frac{P(h \wedge e)}{P(e)}$$

仅考虑 e 为真的那些样本进行计算, 进而决定 h 为真的那些样本在上述样本中所占的比率。舍选采样 (rejection sampling) 的基本思想是像之前一样产生样本, 但是需要舍弃包含 e 为假的那些样本。在剩余的未被抛弃的那些样本中, h 为真的样本所占的比例是 $P(h|e)$ 的一个估测值。在证据是变量赋值的组合条件下, 样本中被赋值的任何一个变量值与观察值不同时, 此样本会被抛弃。

h 的概率的误差取决于没有被舍弃的样本的数目。没有被舍弃的样本的数目与 $P(e)$ 成比例。因此, 在 Hoeffding 不等式中, n 是没有被拒绝的样本的数目。因此, 误差取决于 $P(e)$ 。

259

当证据是不可能时, 舍选采样不会发生作用, 这看起来似乎不是问题, 因为, 通过定义, 不可能的证据不可能发生。虽然对于简单模型来说是真的, 但是对于带有复杂观察值的复杂模型来说, 每一个可能的观察值也许不可能。还有, 对许多应用程序, 例如诊断, 因为涉及异常的观察值时, 用户对确定概率感兴趣。

【例 6-24】 图 6-11 显示了怎样使用舍选采样估测 $P(\text{Tampering} \mid \neg \text{Smoke} \wedge \text{Report})$ 。任何包含 $\text{Smoke} = \text{true}$ 的样本都会被舍弃。在没有考虑任何其他变量的情况下, 样本被舍弃。舍弃包含 $\text{Report} = \text{false}$ 的样本。我们使用剩余样本的样本均值(标记了✓的项)估测关于 Tampering 的后验概率分布。

因为 $P(\neg \text{Smoke} \wedge \text{Report}) = 0.0213$, 我们预期 1 000 个样本中, 有 21 个样本会被舍弃。因此, 21 可以看做是 Hoeffding 不等式中的 n , 也就是说, 例如, 对于任意的概率, 保证 63% 的案例中这些样本的误差小于 0.2。

样本	Tampering	Fire	Alarm	Smoke	Leaving	Report	
s_1	false	true	true	true	×		
s_2	false	false	false	false	false	false	×
s_3	false	true	true	true	×		
s_4	false	false	false	false	false	true	✓
s_5	false	false	false	false	false	false	×
s_6	false	false	false	false	false	false	×
s_7	true	false	false	true	×		
s_8	true	false	false	false	false	true	✓
...							
s_{1000}	true	false	true	true	×		

图 6-11 对于 $P(\text{Tampering} \mid \neg \text{Smoke} \wedge \text{Report})$ 的舍选采样

5. 重要性采样

除了创建一个样本然后舍弃它, 可以将采样和推理混合, 进而推断一个样本被舍弃的概率, 此方法是可行的。在重要性采样中, 每个样本有一个权重, 我们通过使用加权的样本均值来计算样本均值。样本的权值有两个来源。

- 样本并不是以其概率成比例地被选择, 而是根据其他的分布——**建议分布**(proposal distribution)被选择。
- 使用证据更新权值和计算样本被舍弃的概率。

260

【例 6-25】 变量 A 没有父节点, 变量 E 只有一个父节点 A , 但是 A 有其他的子节点。假设 $P(e|a) = 0.03$, $P(e|\neg a) = 0.63$ 且 e 是可以被观察的。考虑包含 $A = \text{true}$ 的那些样本。在 1 000 个这样的样本中, 只有 3 个未被舍弃。除了舍弃 99.7% 的包含 $A = \text{true}$ 的样本, 每个包含 $A = \text{true}$ 的样本可以加权 0.003。因此, 仅有一个样本能够传递关于舍弃的相关信息。

假设 $P(a) = 0.98$ 。如果算法根据概率采样, 在 1 000 个样本中, 有 20 个样本中包含 $A = \text{false}$ 。除了根据概率采样外, 假设从 $A = \text{true}$ 的样本中采样需要 50% 的时间, 但是, 每个样本加权如下。包含 $A = \text{true}$ 的样本加权 $0.98/0.5 = 1.96$, 并且 $A = \text{false}$ 的样本加权 $0.02/0.5 = 0.04$, 很容易可以计算出加权的样本均值和概率是相同的。

在舍选采样中, 对于先前的概率和观察值 e , 在 98% 的样本中 $A = \text{true}$, 并且因为证据 e , 99.7% 的这些样本将会被舍弃。 $A = \text{false}$ 将会被选择在 2% 的样本中, 并且 37% 的这些样本将会被舍弃。因此, 舍选采样接受仅仅 $0.98 \times 0.003 + 0.02 \times 0.63 = 0.01554$ 的样本, 舍弃超过 98% 的样本。

如果将这个思想融入例子的前两段,一半的样本中将包含 $A=true$, 并且这些样本可以加权 $1.96 \times 0.003 = 0.00588$, 并且另外一半的样本中将包含 $A=false$, 它们可以加权 $0.04 \times 0.63 = 0.0252$ 。这样的两个样本可以传递许多舍弃样本的信息。◀

重要性采样和舍选采样在两个方面有不同之处:

- 重要性采样并不对所有的变量进行采样, 只是针对其中的一部分变量。没有被采样的变量和没有被观察到的变量被统计出来(例如, 执行一些精确的推论)。

特别是不想对可观察变量进行采样(然而在算法中, 没有阻止该采样)。如果所有的非观察变量都被采样, 那么很容易可以计算出已知证据的样本的概率(见习题 6.10)。

- 重要性采样不需要根据它们的先验概率对所有的变量进行采样; 它可能使用任意的分布进行采样。对变量采样使用的分布叫做提议分布(proposal distribution)。任意一个分布都可以作为提议分布, 只要该提议分布选择一些样本的概率不为 0, 这在模型中是可能的(否则该部分的空间将不会被利用)。选择一个好的提议分布是很重要的。

一般的, 为了从因式 $f(S)q(S)$ 中统计变量 S , 可以从分布 $q(s)$ 选择样本集合 $\{s_1, \dots, s_N\}$ 。

$$\sum_S f(S)q(S) = \lim_{N \rightarrow \infty} \left(\frac{1}{N} \sum_{s_i} f(s_i) \right) \quad (6.1)$$

261

此等式计算了 $f(S)$ 的期望值, 在这里期望大于了分布 $q(S)$ 值。

在前向采样中, $q(s)$ 是均匀采样, 但是式(6.1)对任意分布都成立。

在重要采样中, S 是将会被采样的变量集合。正如在 VE 算法中, 我们引入一些变量并且将它们统计出来; 在这种情况下, 我们对采样的变量求和:

$$P(h|e) = \sum_S P(h|S, e)P(S|e)$$

顶部和底部同乘以提议分布 $q(S)$, 有

$$P(h|e) = \sum_S \frac{P(h|S, e)P(S|e)q(S)}{q(S)}$$

注意这并未给出一个除数为 0 的错误; 如果 $q(s)=0$, s 将永远不会被选作一个样本。

使用式(6.1), 假设 $\{s_1, \dots, s_N\}$ 是所有样本的集合:

$$P(h|e) = \lim_{N \rightarrow \infty} \sum_{s_i} \frac{P(h|s_i, e)P(s_i|e)}{q(s_i)}$$

使用关于 $P(s_i|e)$ 的贝叶斯规则, $P(e)$ 是一个常数, 有

$$P(h|e) = \lim_{N \rightarrow \infty} \frac{1}{k} \sum_{s_i} \frac{P(h|s_i, e)P(e|s_i)P(s_i)}{q(s_i)}$$

式中, k 是一个标准化的常数, 它保证了对于一个互斥和覆盖假设集合而言, 值的后验概率值渐近到 1。

因此, 对于每个样本, 权值 $P(s_i)/q(s_i)$ 就如同一个先验值乘以证据的概率来获得给定样本的权值。已知多个样本, 前面的公式显示了怎样通过获得每个样本的加权均值来预测关于任意 h 的后验分布。

需要注意重要性采样如何概括舍选采样。舍选采样是 $q(s_i)=P(s_i)$ 的情况, 并且 S 包括所有的变量, 也包含观察变量。

图 6-12 显示了对于已知查询变量 Q 和证据 e , 为了计算 $P(Q|e)$ 所采用的重要性采样算法的详细过程。第一个 for 循环(18 行)显示了创建了关于 S 的样本 s 。变量 p (21 行)表示样本 s 的权值。算法更新了查询变量的每个值的权值, 并且将样本 s 的概率增添到变量

$mass$, 变量 $mass$ 代表概率和(probability mass)——查询变量的所有值的概率的总和。最终, 通过查询变量的每个值的权值除以概率和来返回概率。

```

1: procedure IS_BN( $V_s, P, e, Q, S, q, n$ )
2:   Inputs
3:    $V_s$ : 变量集合
4:    $P$ : 计算条件概率的过程
5:    $e$ : 证据, 为某些变量赋值
6:    $Q$ : 查询变量
7:    $S=\{S_1, \dots, S_k\}$ : 抽样变量集合
8:    $q$ :  $S$  上的分布(提议分布)
9:    $n$ : 产生抽样的数目
10:  Output
11:   $Q$  上的后验分布
12:  Local
13:  数组  $v[k]$ , 这里  $v[i] \in dom(S_i)$ 
14:  实数数组  $ans[m]$ ,  $m$  是  $dom(Q)$  的规模
15:   $s$ :  $S$  的每个元素的赋值
16:   $mass := 0$ 
17:  repeat  $n$  times
18:    for  $i=1$  to  $k$  do
19:      select  $v_i \in dom(S_i)$  使用分布  $q(S_i=v_i | S_0=v_0, \dots, S_{i-1}=v_{i-1})$ 
20:       $s_i =$  赋值  $S_0=v_0, \dots, S_i=v_i$ 
21:       $p_i = P(e | s) \times P(s)/q(s)$ 
22:      for each  $v_i \in dom(Q)$  do
23:         $ans[i] := ans[i] + P(Q=v_i | S \wedge e) \times p$ 
24:       $mass := mass + p$ 
25:  return  $ans[]/mass$ 

```

图 6-12 信念网络推理的重要性采样

【例 6-26】 假设我们想要采用重要性采样来计算 $P(alarm | smoke \wedge report)$ 。必须选择进行采样的变量和提议分布。假设对 *Tampering*、*Fire* 和 *Leaving* 进行采样, 并且使用下面的提议分布:

$$q(tampering) = 0.02$$

$$q(fire) = 0.5$$

$$q(Alarm | Tampering, Alarm) = P(Alarm | Tampering, Alarm)$$

$$q(Leaving | Alarm) = P(Leaving | Alarm)$$

因此, 除了 *Fire* 外, 提议分布和初始分布是一样的。

下面的表给出了几个样本。在这个表中, s 是样本; e 是 $smoke \wedge report$; $P(e | s) = P(smoke | fire) \times P(report | Leaving)$, 其中 *Fire* 和 *Leaving* 的值来源于样本; 当样本中 $Fire=true$ 时, $P(s)/q(s)=0.02$, 且当 $Fire=false$ 时, $P(s)/q(s)=1.98$; $p = P(e | s) \times P(s)/q(s)$ 是样本的权重。

<i>Tampering</i>	<i>Fire</i>	<i>Alarm</i>	<i>Leaving</i>	$P(e s)$	$P(s)/q(s)$	p
false	true	false	true	0.675	0.02	0.0135
true	true	true	false	0.009	0.02	0.00018
false	false	false	true	0.0075	1.98	0.01485
false	true	false	false	0.009	0.02	0.00018

$P(alarm | smoke \wedge report)$ 是包含 $Alarm=true$ 的样本的加权比例。

就精确性怎样依赖于运行时间而言,算法的效率主要取决于:

- 提议分布。为了获得最好的结果,提议分布应该尽可能地接近后验分布。然而,Agent一般不能直接根据后验分布进行采样;如果可以,那么就可以更简单地产生后验概率。
- 对哪些变量进行采样。采样较少的变量意味着每个样本中包含更多的信息,但是每个样本要求更多的时间计算样本的概率。

决定提议分布和对哪些变量进行采样是一种艺术。

6. 粒子滤波

重要性采样一次列举一个样本,针对每个样本,对每个变量进行赋值。也可以始于所有样本,即对每个变量所包含每个样本生成一个值。例如,对于图 6-10,能够产生相同的数据,即通过在产生 *Fire* 值之前产生关于 *Tampering* 的值。**粒子滤波**(particle filtering)算法在移动到下一个变量之前就产生关于一个变量的所有样本。它对变量进行一次扫描,并且对于每个变量扫描一次所有的样本。当动态地产生变量时,算法有一个优势,并且许多变量是不受约束的。它也允许考虑重新采样的新操作。

已知关于某些变量的样本集合, **重新采样**(resampling)需要 n 个样本,每个样本有自己的权值,并且产生 n 个样本的新集合,每个样本有相同的权值。对于一个生成的随机变量,重新采样采用与随机采样相同的方法执行,但是选择的是样本而不是值。一些样本被多次选择,而一些样本则未被选择。

粒子(particle)由变量值的集合和相关联的权值组成。对于给定的证据,命题的概率与命题为真的粒子权值的加权部分成比例。粒子的集合是一个**种群**(population)。

264

粒子滤波是一种采样方法,它起始于一个粒子种群,每个粒子中不对任何变量赋值,其权值为 1,在每一步它能够:

- 选择一个没有被采样或者被总结的变量且这个变量没有被观察过。对于每个粒子,它根据某一提议分布对变量进行采样。粒子权值的更新和重要性采样中的一样。
- 选择一个证据来吸收。在这之前证据应该没有被吸收过。对于给定值的粒子,粒子的权值和证据的概率相乘(总结出相关的和没有被采样的任意变量)。
- 对种群进行重新采样。重新采样通过从种群中选择粒子组建一个新的粒子群,每个粒子群有相同的权值,每个粒子以与其权值成比例的概率被选择。一些粒子被遗忘,一些粒子被重复。

在没有重新采样时,重要性采样和粒子滤波是等价的,但是主要的不同是粒子的产生顺序。在粒子滤波中,针对每个变量,对所有的样本进行采样,然而,重要性采样是在下一个粒子被考虑之前,每个粒子(样本)对所有的变量进行采样。

相对于重要性采样,粒子滤波有两个主要的优势。首先,它能使用无穷数目的变量(后面我们将会看到)。第二,粒子能够很好地覆盖假设空间。然后,重要性采样将会涉及一些低概率的粒子,仅有一些粒子能够覆盖大部分的概率团,重新采样使得一些粒子能够更均匀地覆盖概率团。

【例 6-27】 对于图 6-1 中的信念网络,使用粒子滤波计算 $P(\text{Report} | \text{smoke})$ 。首先,产生粒子 s_1, \dots, s_{1000} 。例如,我们使用给定粒子的正在被采样的变量的条件分布作为提议分布。设想对 *Fire* 进行采样。在 1000 个粒子中,约 10 个粒子包含 $\text{Fire} = \text{true}$ 且约 990 个粒子包含 $\text{Fire} = \text{false}$ (如 $P(\text{fire}) = 0.01$)。它能够解释证据 $\text{Smoke} = \text{ture}$ 。那些包含 $\text{Fire} = \text{true}$ 的粒子可以加权 0.9(如 $P(\text{smoke} | \text{fire}) = 0.9$)并且包含 $\text{Fire} = \text{false}$ 的粒子可

以加权 0.01 (如 $P(\text{smoke} | \neg \text{fire}) = 0.01$)。然后重新进行采样, 每个粒子与其权值成比例地被选择。包含 $\text{Fire} = \text{true}$ 的粒子将以 $990 \times 0.01 : 10 \times 0.9$ 的比率被选择。因此, 约 524 个包含 $\text{Fire} = \text{true}$ 的粒子将会被选择, 其余的粒子包含 $\text{Fire} = \text{false}$ 。依次, 其他的变量被采样直到 *Report* 被采样。

265

注意, 在粒子滤波中粒子并不是独立的, 因此并不能直接应用 Hoeffding 不等式。

6.5 概率和时间

我们对动态系统进行建模, 并将其作为一个信念网络, 这是通过在一个特定的时刻将它的特征作为一个随机变量实现的。首先就状态而言, 我们给出了一个模型, 然后显示了它怎样过渡到特征。

6.5.1 马尔可夫链

马尔可夫链(Markov chain)是一个用于表示值序列的信念网络的特定序列, 如动态系统中的状态序列或者句子中的词序列。

图 6-13 显示了作为信念网络的一个一般的马尔可夫链。网络不必停留在状态 S_4 , 但是它可以无限延伸。信念网络传递了独立性假设

$$P(S_{t+1} | S_0, \dots, S_t) = P(S_{t+1} | S_t)$$

这称为马尔可夫假设(Markov assumption)。



图 6-13 马尔可夫链作为信念网络

通常, S_t 表示 t 时刻的状态。直观的, S_t 传递了能够影响未来状态的所有历史信息。在 S_t 时刻, 你可以看到“对于现在而言, 未来条件独立于过去”。

如果每个时间点的过渡概率是相同的(即, 对任意的 $t > 0$, $t' > 0$, $P(S_{t+1} | S_t) = P(S_{t'+1} | S_{t'})$), 我们就说马尔可夫链是静态的(stationary)。为了详细说明静态马尔可夫链, 必须先说明两个条件概率:

- $P(S_0)$ 指出初始条件。
- $P(S_{t+1} | S_t)$ 说明了动态性, 对于每个 $t \geq 0$, 其值是相同的。

静态马尔可夫链是有意义的, 因为:

- 它们提供了很容易理解的一个简单模型。
- 静态假设通常是自然的模型, 因为特殊领域的动态性并没有随着时间而改变。若动态性随着时间改变, 这是因为能被模型化的其他特征存在。
- 网络能够无限延伸。较少的参数能够给出一个无限的网络。我们可以对未来或过去的任意点进行查询或观察。

为了决定状态 S_t 的概率分布, VE 能够统计以前的变量。注意, S_t 后面的变量和 S_t 的概率不相关且 S_t 后面的变量不需要考虑。这个计算作为一个矩阵乘法被说明, 但是注意到矩阵乘法是 VE 的一个简单形式。类似的, 为了计算 $P(S_i | S_k)$, 其中 $k > i$, 仅 S_k 之前的变量需要考虑。

266

6.5.2 隐马尔可夫模型

隐马尔可夫模型(hidden Markov model, HMM)是包含观察值的马尔可夫链的一个扩

展版本。正如马尔可夫链的过渡状态一样, HMM 也包含状态的观察值。不同状态的观察值部分能够映射到相同的观察值, 并且在不同的时刻相同状态的噪声能够随机映射到不同的状态。

与在马尔可夫链中一样, HMM 下的假设就是在 $t+1$ 时刻的状态仅仅与 t 时刻的状态相关。 t 时刻的观察值仅仅与 t 时刻的状态有关。我们可以使用 t 时刻的变量 O_t 对观察值进行建模, 观察值的域是所有可能观察值的集合。图 6-14 显示了 HMM 的信念网络表示。虽然显示了 4 个阶段的信念网络, 但是它仍然能够无限扩展。

静态的隐形马尔可夫模型包含下面的概率分布:

- $P(S_0)$ 指出了初始条件。
- $P(S_{t+1} | S_t)$ 说明了动态性。
- $P(O_t | S_t)$ 详细说明了概率模型。

HMM 中存在几个常见的问题。

信念网络过滤(filtering)或者监听(monitored)问题就是决定当前的状态, 这个状态依据当前或者先前的观察值。也就是决定

$$P(S_t | O_0, \dots, O_t)$$

注意, 状态 S_t 后的所有状态和观察值是不相关的, 这是因为当计算条件分布时, 它们并没有被观察到或者被忽略。

平滑(smoothing)问题是确定依据未来或者过去的观察值的状态。假设 Agent 已经观察到 t 时刻且此时它想要知道 i 时刻的状态, 其中 $i < k$, 平滑问题就是决定

$$P(S_i | O_0, \dots, O_k)$$

其中当 $i > k$ 时, 所有的变量 S_i 和 V_i 能够被忽略。

定位

设想一个机器人想要根据它的历史行为和检测信号值决定它的位置。此问题称做定位(localization)问题。图 6-15 显示了定位问题的信念网络表示。在时刻 i , 存在变量 Loc_i 、 Obs_i 、 Act_i , 它们分别表示 i 时刻机器人的位置、观察值和行为。在这一部分, 假设机器人的行为能够被观察到(第 9 章节介绍这种情况)。

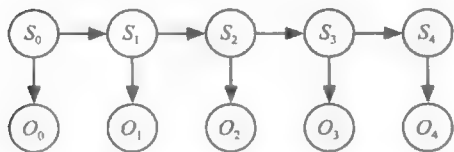


图 6-14 隐马尔可夫模型作为信念网络

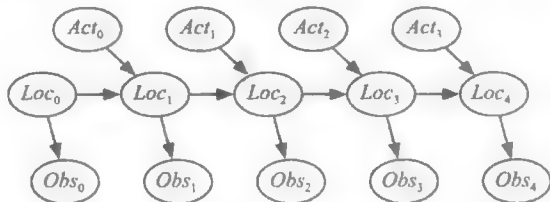


图 6-15 用于定位的信念网络

此模型有如下的动态性假设: 在 i 时刻, 机器人位置在 Loc_i , 它观察到 Obs_i , 然后采取行动, 观察到行为 Act_i , 在 $i+1$ 时刻, 它在位置 Loc_{i+1} 。在 t 时刻, 它的观察值仅仅依赖于 t 时刻的状态。 $t+1$ 时刻机器人的位置依赖于 t 时刻的位置和行为。给出 t 时刻的位置和行为, $t+1$ 时刻它的位置条件独立于以前的位置、以前的观察值、以前的行为。

定位问题即将机器人的位置作为观察历史的函数:

$$P(Loc_i | Obs_0, Act_0, Obs_1, Act_1, \dots, Act_{i-1}, Obs_i)$$

【例 6-28】 考虑到图 6-16 中显示的域。存在一个循环通道, 16 个位置从编号 0 到编号 15。每一时刻, 机器人在其中的一个位置上。将其模型化, 即对于一个时刻 i , 存在一

个变量 Loc_i , 域为 $\{0, 1, \dots, 15\}$ 。



图 6-16 定位域

- 位置 2、4、7 和 11 存在门。
- 机器人有一个传感器，可以识别到它是否在一扇门面前。模型化即对每个时刻 i 。对应一个变量 Obs_i , 域为 $\{door, nodoor\}$ 。假设有下面的条件概率：

$$P(Obs = door | atDoor) = 0.8$$

$$P(Obs = door | notAtDoor) = 0.1$$

其中，在状态 2、4、7、11 时 $atDoor$ 为真，在其他的状态 $notAtDoor$ 为真。

因此，观察值是部分相关的，在许多状态中会出现相同的观察值并且在下面的方式中它是有噪声的：在 20% 的情况下，机器人在一扇门面前，但是传感器却虚假地给出一个消极的信号。在 10% 的情况下，机器人并不在一扇门面前，但是传感器却记录此时存在一扇门。

- 在每个时刻，机器人能够向左移、向右移、静止不动。假设静止不动具有决定性，但是移动的动态性是随机的。它能够实施 $goRight$ 行为并不意味着它走一步就到右边——它可能静止不动，走两步到右边或者以任意的的位置作为终点。对每个位置 L ，假设有以下动态性：

$$P(Loc_{t+1} = L | Act_t = goRight \wedge Loc_t = L) = 0.1$$

$$P(Loc_{t+1} = L + 1 | Act_t = goRight \wedge Loc_t = L) = 0.8$$

$$P(Loc_{t+1} = L + 2 | Act_t = goRight \wedge Loc_t = L) = 0.074$$

$$\text{对任意的其他位置 } L', P(Loc_{t+1} = L' | Act_t = goRight \wedge Loc_t = L) = 0.002$$

所有的定位算法以 16 为模。 $goLeft$ 行为以同样的方式运行。

机器人起初在一个未知的位置，并且必须确定它的位置。

域看起来似乎有歧义性，传感器有噪声，动态性太随机而不能做任何事情。然而，对于给定的行为和观察值的历史记录，计算机器人当前位置的概率是可能的。

图 6-17 给出了机器人位置的概率分布，假设初始时不知道机器人位置的任何信息，并且机器人经历下面的观察值：观察到门，向右走，没有观察到门，向右走，然后，再观察门。位置 4 是最可能的当前位置，拥有 0.42 的后验概率分布。也就是说，就图 6-15 的网络而言：

$$P(Loc_2 = 4 | Obs_0 = door, Act_0 = goRight, Obs_1 = nodoor, Act_1 = goRight, Obs_2 = door) = 0.42$$

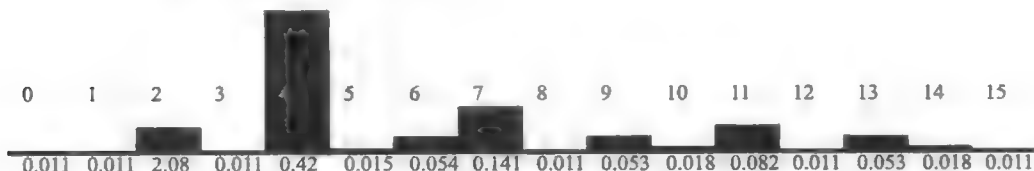


图 6-17 位置的分布。位置从数字 0 到 15 编号。底端的数字指出了当例 6-28 给出特殊的行为序列和观察值序列后，机器人位置的后验概率分布。柱状条的高度和后验概率成比例

位置 7 是第二个可能的当前位置，具有 0.141 的后验概率。位置 0、1、3、8、12 和

15 则是最不可能为当前位置的, 它们具有 0.011 的后验概率。

通过使用书中介绍的网站的小程序可以看看其他的观察序列如何发挥作用。

269

【例 6-29】 利用另一个传感器讨论例 6-28, 假设, 除门传感器外, 还有光传感器。光传感器和门传感器独立于给定的状态。假设光传感器信息性并不是很强, 无论它是否能够侦测到任意的光, 它仅仅能给出是或否的信息, 并且这是有噪声的, 且以位置为基础。

图 6-18 使用了下面的变量进行建模:

- Loc_t 是机器人在 t 时刻的位置。
- Act_t 是机器人在 t 时刻的行为。
- D_t 是 t 时刻的门传感器值。
- L_t 是 t 时刻的光传感器值。

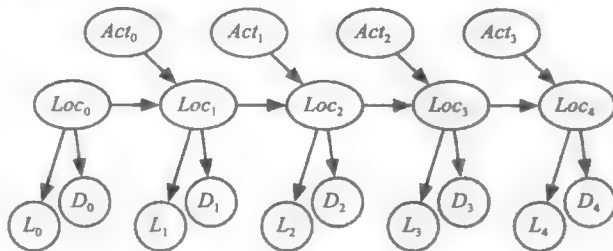


图 6-18 多传感器的定位

在 L_t 和 D_t 的基础上, 结合光传感器和门传感器的信息, 这是一个传感器融合(sensor fusion)的实例。对于给定的信念网络模型, 对传感器融合来说, 定义新的机制不是必要的。标准概率推理结合了两个传感器的信息。

270

6.5.3 监听和平滑算法

为了实施监听和平滑, 可以使用任意一个标准信念网络算法, 例如 VE 或者粒子过滤。利用时间向前移动的事实可以及时获得观察值和有意义的当前状态。

在信念监听(belief monitoring)或者过滤(filtering)中, 给出历史观察值的前提下, 机器人可以计算当前状态的概率。就图 6-14 中的 HMM 而言, 对每一个 i , Agent 想要计算 $P(S_i | o_0, \dots, o_i)$, 其表示就特殊的观察值 o_0, \dots, o_i 而言 i 时刻的状态值, 通过使用 VE 算法可以很容易计算出来:

$$\begin{aligned}
 P(S_i | o_0, \dots, o_i) &\propto P(S_i, o_0, \dots, o_i) \\
 &= P(o_i | S_i) P(S_i, o_0, \dots, o_{i-1}) \\
 &= P(o_i | S_i) \sum_{S_{i-1}} P(S_i, S_{i-1}, o_0, \dots, o_{i-1}) \\
 &= P(o_i | S_i) \sum_{S_{i-1}} P(S_i | S_{i-1}) P(S_{i-1}, o_0, \dots, o_{i-1}) \\
 &\propto P(o_i | S_i) \sum_{S_{i-1}} P(S_i | S_{i-1}) P(S_{i-1} | o_0, \dots, o_{i-1})
 \end{aligned} \tag{6.2}$$

假设 Agent 已经根据得到的 $i-1$ 时刻以前的观察值计算出概率, 也就是说, 它有一个表示 $P(S_{i-1} | o_0, \dots, o_{i-1})$ 的因式。注意到, 这仅仅是关于 S_{i-1} 的因式。为了计算下一个概率, 它将这个因式乘以 $P(S_i | S_{i-1})$, 然后计算出 S_{i-1} , 再乘以因式 $P(o_i | S_i)$, 并且标准化。

将关于 S_{i-1} 的因式乘以因式 $P(S_i | S_{i-1})$, 然后计算出 S_{i-1} 是一个矩阵乘法(matrix multiplication)。将结果乘以 $P(o_i | S_i)$, 这被称为点积(dot product)。矩阵乘法和点积是 VE 的简单实例。

【例 6-30】 考虑到例 6-28 中的域。门的观察值涉及将每个位置 L 的概率乘以 $P(\text{door} | Loc = L)$, 并且重新标准化。对于每个状态的一个向右移动, 是在使用该状态下的概率为状态加权的的过程中, 对右移动作进行的向前模拟。

对许多问题来说, 状态空间太大而不能进行准确的推理。对那些领域来说, 粒子过滤

通常是有效的。对临时模型来说,重新采样通常发生在每一个时间步。一旦观察到证据,且已经计算出样例的后验概率,那么他们就能够被重新采样。

平滑(smoothing)是指在 HMM 中,在给定过去和未来观察值的条件下,计算状态变量的概率分布。未来观察值的使用有助于得到更准确的分布。给定一个新的观察值,通过使用 VE 算法扫描一次所有状态,进而更新所有先前的状态评估,此方法是可能实现的,见习题 6.11。

271

6.5.4 动态信念网络

不必把特殊时刻的状态当做一个单独的变量。用特征表示状态是很自然的。

动态信念网络(dynamic belief network, DBN)是一个带有常规重复结构的信念网络。它就像一个(隐形的)马尔可夫模型,但是状态和观察值用特征表示。假设,时间是离散的。如果 F 是一个特征, F_t 是一个随机变量,其表示变量 F 在 t 时刻的值。一个动态网络包含下面几个假设:

- 每个时刻特征的集合是相同的。
- 对于任意时刻 $t > 0$, 变量 F_t 的父节点是 t 时刻的变量或者是 $t-1$ 时刻的变量,因此图是非周期的。结构并不依赖于 t 值(除 $t=0$ 是一个特殊的时刻)。
- 每个时刻 $t > 0$ 时,变量怎样取决于父节点的条件概率分布是相同的。

因此,一个动态的信念网络中指出了 $t=0$ 时刻的信念网络,每个变量 F_t 指出 $P(F_t | \text{parents}(F_t))$, 在这里, F_t 的父节点是相同的或是前一次步长。对 t 而言,这作为一个自由的参数被指出。对于任意 $t > 0$ 的时刻,可以使用条件概率,正如在信念网络中,直接的循环是不允许的。

动态信念网络模型能够被描述为一个两步信念网络,其表示最初两次(0 和 1)的变量,也就是说,每个特征 F 有两个变量 F_0 和 F_1 , $\text{parents}(F_0)$ 仅仅包括 $t=0$ 的变量, $\text{parents}(F_1)$ 可以是 t 为 0 或者 1 时刻的变量,只要结果图是无环的。和网络相关的概率是 $P(F_0 | \text{parents}(F_0))$ 和 $P(F_1 | \text{parent}(F_1))$ 。因为重复的结构,对每个 $i, i > 1, P(F_i | \text{parents}(F_i))$ 有与 $P(F_1 | \text{parents}(F_1))$ 完全相同的结构和相同的条件概率。

【例 6-31】假设交易 Agent 想要对商品(例如打印纸)交易价格的动态性进行建模。为了表示这个域,设计者对影响价格的变量和其他变量进行了建模。设想纸浆的成本和运输成本直接影响纸的价格。运输成本受天气的影响,纸浆的价格受树虫多少的影响,反过来它也受天气的影响。假设每个变量依赖于先前时间步的值。图 6-19 显示了描述这些依赖关系的两步动态信念网络。

注意,图中的变量初始时是独立的。

这个两步动态信念网络能够被扩展成为一个常规的动态信念网络,这是通过复制每个时间步的节点来实现的,并且未来步中父节点是 1 时刻变量父节点的一个复制。图 6-20 显示了一个扩展的信念网络。下标表示变量相关的时间。

272

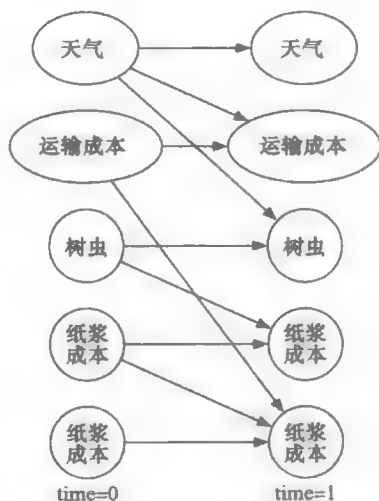


图 6-19 纸张价格的两步动态信念网络

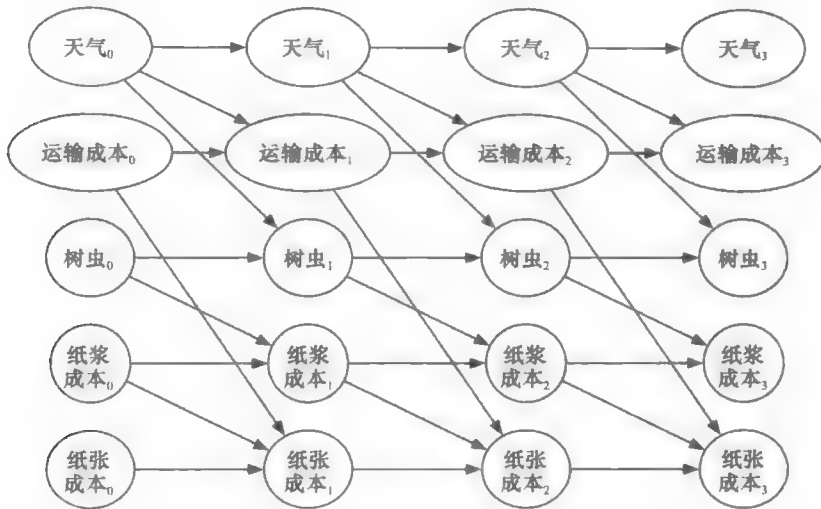


图 6-20 纸张价格的扩展动态信念网络

6.5.5 时间粒度

HMM 的定义和动态信念网络中存在的问题之一是模型依赖于时间粒度。时间粒度可以是固定的，例如一天或者一秒的 $1/30$ ，当然它也可能是基于事件的，此时当一些令人感兴趣的事情发生时，则存在一个时间步。如果时间粒度发生变化，如，从天到小时，那么条件概率也必须发生改变。

独立于时间粒度的一种动态建模方式是针对每个变量和每个变量的值进行建模，即模拟：

- 变量的值预期要保留的时间。
- 当值发生变化时，它将会转变为什么值。

如果给定离散化的时间和状态转移的时间模型，如指数衰减，根据这些信息就可以构建动态信念网络。如果时间的离散化足够精细，忽略每个时间步的多值转移，由此产生的误差会很小。见习题 6.12。

6.6 本章小结

- 概率可以在不确定的情况下用于决策。
- 后验概率基于证据来更新 Agent 的信念。
- 贝叶斯信念网络可以用来描述域的独立性。
- 可以对稀疏图(低树宽)实现精确推理。
- 随机模拟可用于近似推理。
- 隐马尔可夫模型或动态信念网络可用于适时的概率推理，比如定位。

6.7 参考文献及进一步阅读

人工智能的角度介绍概率论和(贝叶斯)信念网络，参考了 Darwiche[2009]、[Koller 和 Friedman, 2009]、Pearl[1988]、Jensen[1996]、Castillo、Gutiérrez 和 Hadi[1996]的介绍。Halpern[1997]综述了逻辑与概率之间的关系。Bacchus、Grove、Halpern 和 Koller[1996]对于概率推理提出了一种随机世界的方法。

274

在 Zhang 和 Poole[1994]、Dechter[1996]的文章提出变量消除的方法来评估信念网络。Bodlaender [1993]探讨了树宽问题。

信息理论的综述可以参阅文献 Cover、Thomas[1991]和 Grünwald[2007]。

对于因果关系的讨论可以参阅文献 Pearl[2000]和 Spirtes 等[2000]。

对于随机模拟的介绍可以参阅文献 Rubinstein[1981]、Andrieu、de Freitas、Doucet 和 Jordan [2003]。信念网络中的前向采样是基于文献 Henrion[1988]的，文中称为逻辑采样。这里所描述的信念网络的重要性采样是基于文献 Cheng 和 Druzdzel[2000]的，文中还考虑到如何学习提议分布。文献 Doucet、de Freitas 和 Gordon[2001]是一系列的关于粒子滤波的文章。

隐马尔可夫模型参见文献 Rabiner[1989]。Dean 和 Kanazawa[1989]介绍了动态贝叶斯网络。Thrun、Burgard 和 Fox[2005]描述了有关概率和机器人学之间关系的马尔可夫定位及其相关问题。定位方面粒子滤波的使用参阅文献 Dellaert、Fox、Burgard 和 Thrun[1999]。

有关人工智能的不确定性，在年会和普通的 AI 会议上会提出最新的研究成果。

6.8 习题

6.1 只使用概率公理和条件独立性的定义证明命题 6.5。

6.2 考虑图 6-21 中的信念网络，其中电气领域延伸到包括高射投影仪。回答以下关于一些变量的值如何影响到另一些变量信念的知识的问题：

(a) “投影仪电源插座接通”的值的知识对“sam 读书”值的信念能产生影响吗？请做出解释。

(b) “屏幕亮”的知识对“sam 读书”的信念有影响吗？请做出解释。

(c) 如果给出你已经观察到“屏幕亮”的值，“投影仪电源插座接通”的知识会对“sam 读书”的信念产生影响吗？请做出解释。

(d) 如果只是观察到“灯正常”，哪些变量可以改变它们的概率？

(e) 如果只是观察到“投影仪通电”，哪些变量可以改变它们的概率？

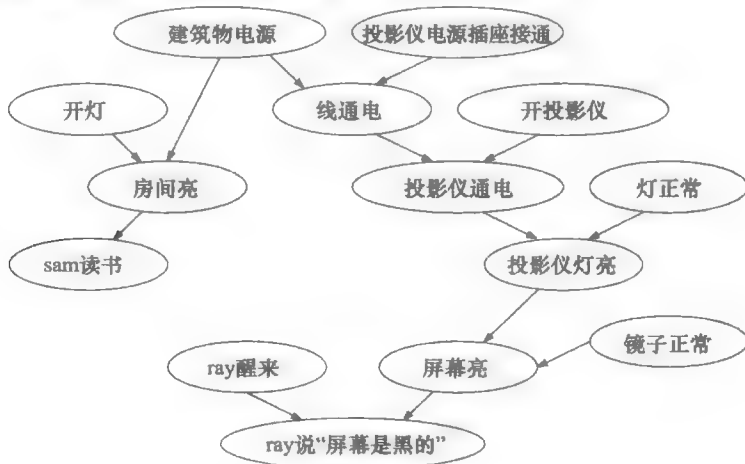


图 6-21 高射投影仪信念网络

6.3 使用信念网络表示习题 5.8 中的情形。显示网络结构并隐藏观察到的节点。给出所有初始因式，对于条件概率作出合理的假设(应该遵循习题中的情节，但允许一些噪音)。

6.4 假设要诊断出当加入多位二进制数时在校学生所犯的错误的。假设只考虑增加两个两位数字，以形成一个三位数字。

275

也就是说，问题的形式为：

$$\begin{array}{r} A_1 A_0 \\ + B_1 B_0 \\ \hline C_2 C_1 C_0 \end{array}$$

其中, A_i 、 B_i 和 C_i 是所有二进制数字。

(a) 假设我们想要模拟学生是否知道二进制加法以及他们是否知道如何运算。如果他们知道, 他们通常会得到正确的答案, 但有时他们会犯错误。如果他们不知道如何做相应任务, 他们就只能猜测。

你必须指出建立二进制加法模型, 哪些变量是必需的, 哪些错误是学生会的, 也就是说, 每个变量代表什么。给出一个 DAG 指定这些变量依赖关系。

(b) 对域来说, 什么是合理的条件概率?

(c) 可以通过使用 Alspace.org 信念网络工具来实现。测试不同情况下你的表示。

给出图来解释每个变量的含义、给出概率表, 并且展示各种例子如何运作。

6.5 在这个问题中, 你要构建一个在习题 5.10 描述的深空一号飞船(DS1)的信念网络表示。图 5-14 描述了实际 DS1 发动机设计的一部分。

276

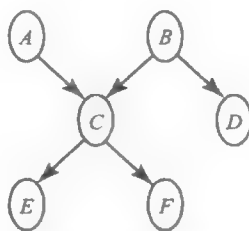
假设以下情形:

- 阀门可能是 *open* 或 *closed*。
- 如果阀门处于打开状态, 气体会流出, 如果阀门处于关闭状态气体不会流出, 此时它的值都为 *ok*; 气体永远都不会流出的情况下, 它的值为 *broken*; 不论阀门是打开还是关闭, 气体都会独立流出, 它的值为 *stuck*; 或者如果气体从阀门泄漏而不是流入, 它的值是 *leaking*。
- 有三个气体传感器能够探测到气体泄漏(但不会探测出是哪种气体)。第一个气体传感器检测来自最右边的阀门($v1, \dots, v4$)中的气体, 第二个气体传感器检测来自中间阀门($v5, \dots, v12$)的气体, 第三个气体传感器检测来自最左边阀门($v13, \dots, v16$)的气体。

(a) 建立一个域的信念网络表示。必须考虑到最上面的阀门(那些送入发动机 $e1$ 的阀门), 确保有适当的概率。

(b) 在一些特殊例子中测试你的模型。

6.6 考虑下面的信念网络:



其布尔变量(将 $A=true$ 记为 a , 将 $A=false$ 记为 $\neg a$)和条件概率如下:

$$\begin{array}{ll}
 P(a) = 0.9 & P(d|b) = 0.1 \\
 P(b) = 0.2 & P(d|\neg b) = 0.8 \\
 P(c|a, b) = 0.1 & P(e|c) = 0.7 \\
 P(c|a, \neg b) = 0.8 & P(e|\neg c) = 0.2 \\
 P(c|\neg a, b) = 0.7 & P(f|c) = 0.2 \\
 P(c|\neg a, \neg b) = 0.4 & P(f|\neg c) = 0.9
 \end{array}$$

(a) 使用 VE 来计算 $P(e)$ 。首先, 删除无关变量。对于给定的消除顺序显示创建的因式。

(b) 假设使用 VE 来计算 $P(e|\neg f)$ 。以前计算有多少可重复利用? 显示出不同于(a)部分中的因式。

6.7 解释如何拓展 VE 来允许更多的普通观察和查询。特别的, 请回答以下问题:

- 如何拓展 VE 算法以允许变量值的析取(例如, $X=a \vee X=b$ 这种格式)?
- 如何拓展 VE 算法以观察不同变量值的析取(例如, $X=a \vee Y=b$ 这种格式)?
- 如何拓展 VE 算法来允许计算一组变量的边缘概率(例如, 求解 $P(XY|e)$ 的边缘概率)?

277

6.8 在核研究潜艇中, 传感器测量反应堆堆芯的温度。如果传感器读数异常高($S=true$), 表明核心过

热($C=true$), 就触发报警($A=true$)。报警器或传感器有缺陷($S_{ok}=false, A_{ok}=false$), 这可能会导致它们出现故障。可以通过图 6-22 中的信念网络来建立报警系统的模型。

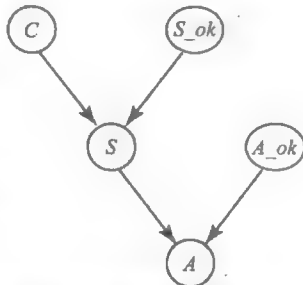


图 6-22 核潜艇的信念网络

(a) 此网络中的初始因式是什么? 对于每个因式, 列出它代表什么, 它是什么变量的函数。

(b) 给出报警器无故障, 即 $P(c | \neg a)$ 的情况下如何使用 VE 来计算核心过热的概率。对于每个消除的变量, 说明哪些变量被消除, 哪些因式被移除, 又建立了哪些因式, 包括每个因式作用在什么变量, 解释如何通过最后一个因式获得答案。

(c) 假设我们又添加一个相同的传感器到系统中, 当两个传感器中任何一个读到高温时就触发报警器。两个传感器坏掉或是读取失败都是相对独立的。给出相应的扩展信念网络。

6.9 继续习题 5.14。

(a) 解释一个信念网络模型需要的知识(关于物理的和学生的)?

(b) 在域中使用信念网络(使用诱导性诊断或基于一致性的诊断)的主要优势是什么?

(c) 在此域中与使用信念网络相比, 使用诱导性诊断或基于一致性诊断的主要优势是什么?

278

6.10 在重要性采样中, 采样每个非观察变量, 不需要全面执行 VE 算法。在这种情况下给出证据, 解释如何计算采样的概率。【提示: 注意抽取子节点以及可观察变量的父节点都是可能的。】

6.11 考虑隐马尔可夫模型中的滤波问题。

(a) 对于某给定未来与过去观察结果的变量 X_j , 给出概率公式。公式应包含一个来自先前状态的因式, 一个来自后续状态的因式, 它们相结合决定 X_k 的后验概率。 X_{j-1} 需要的因式在没有重新计算来自 X_{j+1} 的信息下, 是如何计算的? 【提示: 考虑删除最左边的变量, 也删除最右边的变量, 如何用 VE 来计算 X_j 的后验分布。】

(b) 假设已计算出每个状态 S_1, \dots, S_k 的概率分布, 然后得到时刻 $k+1$ 的观察。每个变量的后验概率在时间 k 是如何线性更新的? 【提示: 需要存储的不仅是每个 S_i 上的分布。】

6.12 考虑产生动态信念网络的问题。正如第 6.5.5 节中一样, 给定一个特定的离散化时间、过渡时间和状态转移的表示。假设存在一个变量在一个状态中能保持时间的指数分布, 并且指定了每个变量值的半衰期。给出动态信念网络的表示, 假设每一时间步内仅有一个过渡。

6.13 假设你的老板对背着相机、绕着工厂的机器人定位感兴趣。老板听过变量消除、舍选采样、粒子滤波的知识, 想知道这些技术中哪个最适合此项任务。你必须给老板写一份报告(使用适合的语句), 说明哪种技术是最适合的。解释为什么放弃另外两种技术。对于最合适的技术, 说明使用这种技术定位必需的信息是什么。

(a) VE(即隐马尔可夫模型中的精确推理)。

(b) 舍选采样。

(c) 粒子滤波。

279

}

280

第三部分

Artificial Intelligence: Foundations of Computational Agents

学习与规划

学习概述与有监督学习

年轻时荒废学业的人，失去的不仅是过去的时光，还有他的未来。

——Euripides(公元前 484—前 406，古希腊悲剧作家)

《佛里克索斯》，Frag. 927

学习指的是一个 Agent 基于经验改善其行为的能力。这意味着 Agent 能够：

- 扩展活动的范围——做得更多。
- 提高任务的精确性——做得更好。
- 提高处理速度——做得更快。

学习是一个智能 Agent 的必备能力。正如 Euripides 指出的那样，学习涉及一个 Agent 以某种方式记住在未来能派上用场的过去经验。

本章介绍有监督学习：给定一组由输入-输出对组成的训练样本，预测一个新输入所对应的输出。我们将说明这种学习方式可由 4 种途径加以实现：选择单一的完美适应训练样本的假设；从训练样本直接做出预测；选择与训练样本一致的假设空间的子集；寻找条件依赖于训练样本的假设的后验概率分布。

第 11 章介绍有监督之外的学习模型，包括聚类(经常被称做无监督学习)、概率模型与增强学习。14.2 节介绍关系表示学习。

283

7.1 学习问题

任一个学习问题均由下述三个部分组成：

任务(task)：欲改善的行为或任务。

数据(data)：用于在任务中提升性能的经验。

性能改善度量(measure of improvement)：性能改善是如何度量的？例如，获得了最初不具备的新技能，提高了预测的准确率，或是提高了处理的速度等。

考虑如图 2-9 所示的 Agent 内部构造，学习就是输入先验知识和数据(例如，有关 Agent 经验的数据)并形成一种内部的表示形式(知识库)，这种表示可以被 Agent 在行动时加以利用。

这种内部表示可以是原始的经验数据本身，但更为常见的是对数据进行概括的简洁表示。基于样例推断内部表示的过程经常被称为**归纳(induction)**。与之对应的一个概念是演绎，它从知识库经由逻辑推理得到结论；与之对应的另一个概念是溯因推理，它假定对于一个特例来说什么可能是正确的。

在选择表示形式的时候，存在两种相互矛盾的准则：

- 表示形式越丰富，对后续的问题求解就越有用。对 Agent 学习问题求解方法来说，表示形式必须足够丰富以便能够表达这种求解方法。
- 表示形式越丰富，学习就越困难。非常丰富的表示形式难于进行学习的原因是它需要大量的数据，而且经常存在不同的假设与这些数据一致。

智能所需的表示形式是两种准则之间的一种折中(见 1.4 节)。学习表示形式的能力是智能的一种,但绝不是唯一的一种。

学习技术包括如下一些议题:

任务: 实际上, Agent 能够获得数据或经验的任意任务均可学习。最常见的学习任务是**有监督学习**(supervised learning),即给定一些输入特征、一些输出特征和一个训练集(training example)(其中的每个样例都指定了输入特征和输出特征的值),预测一个新样例(其输入特征值已知)的目标特征值。当目标变量取离散值的时候,这种学习称为**分类**(classification);而当目标变量取连续值的时候,则称为**回归**(regression)。

其他学习任务包括:训练样本类别未知时的分类学习(无监督学习)、基于奖惩的学习(增强学习)、学习更快推理(分析学习),以及学习更为丰富的表示形式如逻辑程序(归纳逻辑编程)或贝叶斯网络。

反馈: 学习任务可以通过施加于学习器的反馈来描述。在**有监督学习**中,用于学习的每个样例均已被指定了类别。当一个训练器给出每一个样例的分类时就产生了监督分类。当 Agent 获得了对于每个动作(活动)的价值的即刻反馈时就产生了动作(活动)的监督学习。当没有给定类别信息,学习器必须自己发现数据中的类别和规则时发生**无监督学习**(unsupervised learning)。反馈经常发生于这两种极端情况(即有监督学习与无监督学习)之间,例如在**增强学习**(reinforcement learning)中,基于奖惩的反馈发生在一系列动作之后。这会带来所谓的“**信任度分配问题**”(credit-assignment problem),即确定动作的奖赏或惩罚问题。举个例子,用户可能奖励一个传送机器人但并没有确切告知是因为什么而获奖,此时机器人就必须学习获奖的原因,即在什么条件下执行什么动作。在没有明确哪些动作的后果会带来奖赏的情形下,学习执行什么动作是可能的。

表示: Agent 的经验必然影响其内部表示形式。许多机器学习方法的研究就是基于特定表示形式(决策树、神经网络或案例等)的。本章介绍一些标准的表示形式以展示学习的共同特征。

在线学习与离线学习: 在**离线学习**(offline learning)中,所有的训练样本在 Agent 行动之前就已经给定。而在**在线学习**(online learning)中,训练样本在 Agent 行动中到达。这需要 Agent 在它观察到所有的样本之前就具有某些关于已经观察到的样本的表示的知识,当一个新样本来临时更新这种表示。总之,Agent 从没看到所有的样例。**主动学习**(active learning)是在线学习的一种形式。在这种学习中 Agent 通过行动获得对学习有用的样例,即 Agent 推断哪些样例是对学习有用的并收集这些样例。

度量: 学习基于某些提升性能的度量来定义。为了知道 Agent 是否成功学习,必须定义一些学习成功与否的度量。这种度量往往不是考察 Agent 在训练样本上的表现好坏,而是对于新样本它表现如何。

在分类学习中,能够正确分类所有的训练样本并非问题的本质所在。例如,考虑一个给定样本集的布尔特征预测问题,并假设有两个 Agent,即 P 和 N 。 P 认为属于负类的样本就是训练集中的负样本,其他样本(包括不在训练集中的样本)均为正样本; N 则认为属于正类的样本就是训练集中的正样本,其他样本(包括不在训练集中的样本)均为负样本。显然,它们都正确地分类了训练集中的每一个样本,但对于其他样本的分类则是相互矛盾的。学习是否成功,不能看正确分类了训练样本与否,而是要看是否能够正确分类还未出现的样本。因此,一个学习器必须具有**泛化**(generalize)能力:超越特定的训练样本去分类还未出现的样本。

285

度量成功与否的一种标准方法是将样本分为训练集和测试集，利用训练集建立表示，然后在测试集上考察其预测准确率。当然，这只是我们所需要的度量的一种近似，真正的度量是它在未来任务上的表现能力。

偏置：对于一个假设的偏好称为偏置(bias)。考虑先前定义的 Agent N 和 Agent P ，我们说一个假设优于 N (或 P)的假设，不是指从训练集(在该集合上 N 和 P 均准确地分类了所有的样本)上得到的某些结论，而是指从训练集之外的数据上获得的结果。如果没有偏置，Agent 将无法对未知的样本做出预测。 P 和 N 采用的假设对于未来样本的预测是互相矛盾的，如果 Agent 不能从中选择一个较好的假设，它将不能消除这种分歧。总之，为了使得任意的归纳过程对未来的样本能够做出预测，Agent 需要一个偏置。什么是一个好的偏置是一个经验性的问题，它考察什么样的偏置在实践中工作得最好。对于 P 或 N 的偏置，我们没有认定它们在实践中工作良好。

搜索学习：给定表示形式和偏置，学习可以简化为一个搜索问题。在可能的表示空间中进行搜索，以便在给定偏置的条件下找出与数据最匹配的表示形式。不幸的是，除了最简单的样例，搜索空间往往很大，以致进行完全搜索是不可行的。在机器学习中，几乎所有的搜索技术都可以被看成是在表示空间上的局部搜索。学习算法可由搜索空间、评估函数和搜索方法来定义。

噪声：大多数现实世界中的情况是数据并非完全正确。数据中存在噪声(一些特征被赋予错误的值)、不恰当的特征(该特征于预测分类无益)以及带有特征缺失的样本。学习算法的一个重要特征是其处理噪声数据的能力，不管这种噪声是以何种形式存在。

插值和外推：对于那些天然就有“在……之间”插值的情形，例如对时间或空间进行预测，插值在拥有数据的案例之间做预测。外推指的是对还未出现的样本做预测。一般的，外推比插值更不精确。例如，在古代天文学，托勒密体系(天动说)和日心体系(地动说)均用周转圆(圆里套着圆)对太阳系的运动进行了细致的建模。模型的参数可以调整到与数据非常匹配，从而使得该模型在插值方面做得非常好，但在外推方面却是非常糟糕。再举一个例子，在给定某一天之前及之后的价格数据的情况下，预测该天的股票价格通常是容易的，但要预测明天的股票价格却非常困难，而这种预测未来的能力又是极其有价值的。Agent 必须对这种情形小心翼翼，测试案例大多涉及数据点之间的插值，但学得模型却用于外推。

286

为什么我们信任一个归纳的结论

从数据中学习时，Agent 对超越给定的数据做预测。观察到太阳每天升起，人们就预测太阳明天也将升起；观察到没有支撑的物体反复降落，小孩就推断它将总是降落(直到他偶然看到充满氦气的气球)；观察到许多天鹅是黑色的，某些人就可能断定所有的天鹅都是黑色的。考虑如图 7-1 所示的数据，学习算法确定一种表示形式以便预测用户的行为——在该用户所属的样例中，*Author*、*Thread*、*Length* 和 *WhereRead* 字段(特征)的值分别是 *unknown*、*new*、*long* 和 *work*，但没有告知用户将如何行动。这个例子带来的问题是，Agent 为何要相信这种并非逻辑推导(基于 Agent 的知识)所得到的结论？

当 Agent 采用了一个偏置或选择了一个假设，它就置身于训练数据之外了——即使要预测的新样例与数据中的旧样例完全一致。为什么 Agent 相信这个假设而非其他假设？Agent 根据什么标准选择假设？

最常用的技术是依据**奥卡姆剃刀原理**(Ockham's razor)选择适合数据的最简单的假设。威廉·奥卡姆是一位英国哲学家,他出生于大约1285年,1349年死于一场瘟疫。(注:“Occam”是英国小镇“Ockham”的法语拼写,这种写法经常被采用。)奥卡姆在解释经济现象时主张“如无必要,勿增实体”[Edwards, 1967, Vol. 8, P. 307]。

为什么我们信任最简单的假设,特别是为什么要根据表达假设的语言来判定哪个假设是最简单的?

首先,可以合理地假定现实世界是存在结构的,Agent需要发现这种结构以便做出合适的行动。搜索是发现现实世界结构的一种合理方法。高效的搜索策略是从简单的假设开始,然后才是复杂的假设。如果没有发现结构,那将一事无成!现实世界中已经发现了许多结构的事实(例如物理学家发现的所有结构)让我们相信搜索是有效的。

简洁性依赖于语言不会令人生疑。语言在不断地发展进化,它使得人们能够表达现实世界的结构。因此,我们期望日常语言中的简洁性是复杂性的一个很好的度量。

信任归纳假设的最重要原因是信任它们会带来好处。它们帮助Agent与世界进行交互和避免被杀害;一个Agent如果学习基础不牢靠,它将不能长期存活。“最简单假设”这种启发式规则是有用的,因为它在实际中奏效可行。

287

7.2 有监督学习

有监督学习的抽象定义如下。假定学习器获得了下述数据:

- 一个输入特征(input feature)的集合, X_1, \dots, X_n ;
- 一个目标特征(target feature)的集合, Y_1, \dots, Y_k ;
- 一个训练样本(training example)的集合, 对于每一个样本, 其输入特征的值和目标特征的值均已给定;
- 一个测试样本(test example)的集合, 其中只给定了输入特征的值。

学习的目标是预测测试样本和尚未出现的样本中目标特征的值。典型的, 学习是表示形式的构建, 该表示形式可依据新样本输入特征的描述进行预测。

令 e 、 F 和 $val(e, F)$ 分别表示一个样本、一个特征和样本 e 在特征 F 上的值。

【例 7-1】 图 7-1 所示的是一个典型分类任务的训练与测试样本。目标是预测某个人是否阅读发布在电子公告牌上的文章(已经给定该文章的特性)。输入特征分别是 *Author*、*Thread*、*Length* 和 *WhereRead*; 目标特征只有一个, 即 *UserAction*。总共有 18 个训练样本, 它们的特征值均已知。

在该数据集中, $val(e_{11}, Author) = unknown$, $val(e_{11}, Thread) = followUp$, $val(e_{11}, UserAction) = skips$ 。

学习目标是预测给定了输入特征值的新样本的 *UserAction* 值。

最常用的学习方法是考察包括所有可能表示的**假设空间**(hypothesis space)。每一个可能的表示均为一个**假设**(hypothesis)。假设空间通常是有限大, 或是可列无限大的空间。可应用下述方法进行预测:

- 根据某个较好的度量, 在假设空间中找到最好的假设;
- 所有假设均与训练样本一致;
- 在训练样本提供证据的前提下, 计算假设的后验概率。

一种例外的情形是, 基于案例的推理直接使用样本。

示例	Author	Thread	Length	WhereRead	UserAction
e_1	known	new	long	home	skips
e_2	unknown	new	short	work	reads
e_3	unknown	follow Up	long	work	skips
e_4	known	follow Up	long	home	skips
e_5	known	new	short	home	reads
e_6	known	follow Up	long	work	skips
e_7	unknown	follow Up	short	work	skips
e_8	unknown	new	short	work	reads
e_9	known	follow Up	long	home	skips
e_{10}	known	new	long	work	skips
e_{11}	unknown	follow Up	short	home	skips
e_{12}	known	new	long	work	skips
e_{13}	known	follow Up	short	home	reads
e_{14}	known	new	short	work	reads
e_{15}	known	new	short	home	reads
e_{16}	known	follow Up	short	work	reads
e_{17}	known	new	short	home	reads
e_{18}	unknown	new	short	work	reads
e_{19}	unknown	new	long	work	?
e_{20}	unknown	follow Up	long	home	?

图 7-1 用户阅读偏好的例子。这是一些通过观察用户决定是否阅读发布到有线状图案装饰的讨论板上的文章而收集到的训练和测试样本。用户的决定依赖于文章的一些特性，如作者是否知名、是否开始一个新主题或只是跟随式的讨论、文章的长度和在家阅读还是在单位阅读。 e_1, \dots, e_{18} 为训练样本。目标是预测样本 e_{19} 、 e_{20} 以及目前还未出现的样本上用户的行为(特征 *UserAction* 的值)

7.2.1 评估预测

如果 e 为一个样本，目标特征 Y 的点估计(point estimate)指的是 Y 在样本 e 里的预测值，记为 $pval(e, Y)$ 。 e 在 Y 上的误差(error)度量 $pval(e, Y)$ 和 $val(e, Y)$ 的接近程度，其中 $val(e, Y)$ 指的是 Y 在 e 里的真实值。

对于回归问题，目标特征 Y 取实数(连续)值，因而 $pval(e, Y)$ 和 $val(e, Y)$ 均为实数，它们可以进行算术意义上的比较运算。

对于分类问题，目标特征 Y 取离散值，这可分为多种情况加以讨论：

- Y 是二元变量(即取值为 0 或 1)，预测值是某个实数。此时预测值和实际值可以进行数值比较。
- Y 是多元变量(即取值多于二值)，且有时所取的值具有全序关系、大小可按比例进行缩放(使得 Y 的每一个取值都可以用一个实数进行关联)。在这种情况下，预测值和真实值可以基于比例关系做比较。然而，这种方法即使是在取值具有全序关系的条件下也经常是不合适的。例如，假定取值范围是 *short*、*medium* 和 *long*，预测值为 *short* \vee *long* 和预测值为 *medium* 是显然不同的。
- Y 是多元变量，取值范围为 $\{v_1, \dots, v_k\}$ ， $k > 2$ ，此时可以为每个 v_i 单独做预测。这可以通过引入一个关联于每个 v_i 的二元指示变量(indicator variable)来实现。对于每一个样本，当 Y 取值 v_i 时指示变量取值 1，否则取值 0。预测时给出 k 个实数——每个实数对应一个 v_i 。

【例 7-2】 假定一个交易 Agent 想学习一个人对于假期长度(1、2、3、4、5 或 6 天)的偏好。

一种表示形式是定义一个取值范围为假期天数的实值变量 Y 。

另一种表示形式是定义 6 个实值变量, Y_1, \dots, Y_6 , 其中 Y_i 代表想休假 i 天。对于每一个样本, 当假期天数是 i 时 $Y_i=1$, 否则 $Y_i=0$ 。

下面是对包含 5 个数据点的样本集采用两种不同表示方法的例子:

示例	Y	示例	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6
e_1	1	e_1	1	0	0	0	0	0
e_2	6	e_2	0	0	0	0	0	1
e_3	6	e_3	0	0	0	0	0	1
e_4	2	e_4	0	1	0	0	0	0
e_5	1	e_5	1	0	0	0	0	0

在第一种表示方法中, 对一个新样本的预测值可以是任意实数, 例如 $Y=3.2$ 。

在第二种表示方法中, 学习器将预测每个 Y_i 的值。一组可能的预测值是 $Y_1=0.5$, $Y_2=0.3$, $Y_3=0.1$, $Y_4=0.1$, $Y_5=0.1$, $Y_6=0.5$, 它表明这个人可能愿意休假 1 天或 6 天, 而不是 3、4 或 5 天。

在下面的定义中, E 代表所有样本的集合, T 代表目标特征的集合。

关于预测的一些度量可定义如下:

- E 上的绝对误差 (absolute error) 指的是在每个样本上做预测所得的绝对误差之和, 即

$$\sum_{e \in E} \sum_{Y \in T} |\text{val}(e, Y) - \text{pval}(e, Y)|$$

绝对误差总是非负的, 仅当预测值与观察的实际值完全一致时才为 0。

- E 上的平方和误差 (sum-of-squares error) 定义为

$$\sum_{e \in E} \sum_{Y \in T} (\text{val}(e, Y) - \text{pval}(e, Y))^2$$

290

与小误差相比, 该度量更强调大误差的影响: 大误差是小误差的两倍, 导致的恶劣影响是 4 倍大; 大误差是小误差的 10 倍, 导致的恶劣影响是 100 倍大。

- E 上的最坏情况误差 (worst-case error) 指的是样本中的最大绝对误差:

$$\max_{e \in E} \max_{Y \in T} |\text{val}(e, Y) - \text{pval}(e, Y)|$$

此时, 学习器通过其最差的情形来评估其性能。

【例 7-3】 假定一个实值目标特征 Y 由一个实值输入特征 X 决定, 包含如下数据点 (X, Y) :

$(0.7, 1.7), (1.1, 2.4), (1.3, 2.5), (1.9, 1.7), (2.6, 2.1), (3.1, 2.3), (3.9, 7)$

图 7-2 显示了训练数据 (实心圆) 和三条直线 P_1 、 P_2 、 P_3 (它们根据 X 值预测 Y 值)。

P_1 、 P_2 和 P_3 分别是最小化样本绝对误差、平方和误差与最坏情况误差的线性 (函数)。

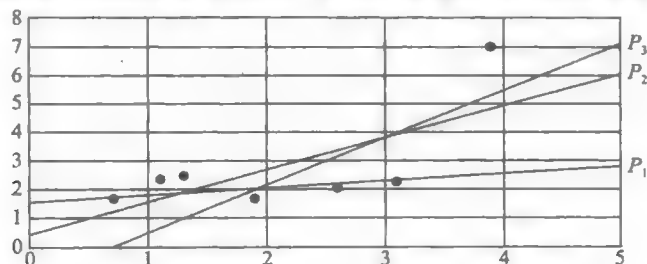


图 7-2 对一个简单样例集的线性预测。实心圆代表训练样本。 P_1 、 P_2 和 P_3 分别是最小化样本绝对误差、平方和误差与最坏情况误差的线性预测 (函数)。参见例 7-3

在 $X=1.1$ 处, P_1 和 P_2 给出了相似的预测值, 即 P_1 是 1.805, P_2 是 1.709; 而训练数据包含的数据点是 (1.1, 2.4)。同时, P_3 给出的预测值为 0.7。在 $[1, 3]$ 之间进行插值, P_1 、 P_2 和 P_3 给出的预测值的差别不超过 1.5。而当进行外推的时候, 预测值的偏差就越来越明显了, P_1 和 P_3 在 $X=10$ 处的预测值是非常不同的。

最小化不同误差度量的直线的差别主要在于它们处理异常样本的不同。数据点 (3.9, 7) 为异常样本, 而其他数据点近似地成一直线。

291

基于最坏情况误差的预测直线 P_3 仅依赖于 3 个数据点, 即 (1.1, 2.4)、(3.1, 2.3) 和 (3.9, 7)。对于 P_3 而言, 这些数据点具有相同的最坏情况误差值。其他数据点可以处于不同的位置, 只要它们偏离 P_3 的距离比这 3 个数据点小。

相比之下, 最小化绝对误差的预测直线 P_1 并不随训练样本的真实 Y 值的变化而变化: 直线之上的点依然在直线之上, 直线之下的点依然在直线之下。例如, 即使最后一个数据点是 (3.9, 107) 而不是 (3.9, 7), 直线 P_1 也保持不变。

最小化平方和误差的预测直线 P_2 对所有的数据点都很敏感: 如果改变任一数据点的 Y 值, P_2 也将随之改变。

对于一种特殊的情形, 即 Y 取值 $\{0, 1\}$ 、预测值是 $[0, 1]$ 之间的某个实数, 则存在一些用于布尔域的预测度量标准, 其中 *true* 和 *false* 分别被看成 1 和 0。

- **数据的似然性** (likelihood of the data) 指的是当预测值被解释为概率的时候数据取值的概率:

$$\prod_{e \in E} \prod_{Y \in T} pval(e, Y)^{ml(e, Y)} (1 - pval(e, Y))^{(1 - ml(e, Y))}$$

$val(e, Y)$ 和 $(1 - val(e, Y))$ 两者之中一个为 1, 另一个为 0。因此, 当 $val(e, Y)=1$ 时该乘积使用 $pval(e, Y)$, 否则使用 $(1 - pval(e, Y))$ 。具有较大似然性的预测被认为是较好的, 而具有最大似然性的模型则称为**最大似然模型** (maximum likelihood model)。

- 基于将 $pval(e, Y)$ 视为概率的编码规则, 数据的**熵** (entropy) 指的是用于编码数据的二进制位数:

$$- \sum_{e \in E} \sum_{Y \in T} [val(e, Y) \log pval(e, Y) + (1 - val(e, Y)) \log (1 - pval(e, Y))]$$

较好的预测指的是具有较小熵的预测。

由于熵是似然性的对数的负数, 所以最小化熵和最大化似然性是等价的。

- 在 Y 取值 0 或 1 的基础上, 限定预测值也取值 0 或 1。**假正错误** (false-positive error) 指的是错误的正预测 (即预测值是 1, 而真实值却是 0)。**假负错误** (false-negative error) 指的是错误的负预测 (即预测值是 0, 而真实值却是 1)。不同的错误通常与不同的代价相关联, 例如如果存在关于产品是否安全的数据, 那么在产品不安全的情形下断言产品是安全的代价与在产品安全的情形下断言产品是不安全的代价可能是不同的。

我们可以根据 Agent 对于学习之外的样本的预测的好坏来判定 Agent 是否拥有一个好的学习算法。Agent 进行预测存在两种极端情况: 当它确信预测是正的时候, 则仅给出正的预测; 除非它确信预测可能是负的, 否则可以给出正的预测。通常的预测介于两者之间。

292

考察独立于决策的预测的一种方法是考虑下述基于预测值和真实值的 4 种指标:

	实际为正	实际为负
预测为正	真正 (tp)	假正 (fp)
预测为负	假负 (fn)	真负 (tn)

假定 tp 是预测为正且实际也为正的样本个数, fp 是预测为正而实际却为负的样本个数, fn 是预测为负而实际却为正的样本个数, tn 是预测为负且实际也为负的样本个数。**精确度**(precision)定义为 $\frac{tp}{tp+fp}$, 即预测为正且实际也为正的样本个数占预测为正的样本的比例。**召回率**(recall)或**真正率**(true-positive rate)定义为 $\frac{tp}{tp+fn}$, 即预测为正且实际也为正的样本个数占实际为正的样本的比例。**假正率**定义为 $\frac{fp}{fp+tn}$, 即预测为正而实际为负的样本个数占实际为负的样本的比例。

Agent 应该尽力最大化精确度和召回率、最小化假正率, 然而, 这些目标是不相容的。Agent 可以仅通过预测为正(如果它确信的话)来最大化精确度和最小化假正率, 但是这样做降低了召回率。反之, 最大化召回率的预测也是有风险的, 即降低了精确度和增大了假正率。Agent 通常具有一些参数, 它们通过阈值的变化控制何时进行为正的预测。**精确度-召回率曲线**(precision-recall curve)绘制了当这些参数变化时的精确度与召回率的相对变化。ROC(receiver operating characteristic, 接收者操作特性)曲线绘制了当这些参数变化时的假正率与假负率的相对变化。上述的每一种方法都可用于学习算法的比较, 这些算法并不依赖于 Agent 的真正意志。

- 可以将预测看成是 Agent 的一个行动。Agent 应该选择最大化偏好函数的行动, 该偏好函数需要在不同行动所关联的代价之间进行折中。行动可能比“真”或“假”这种情况更为复杂, 例如“谨慎行事”或“绝对真实”。第9章将讨论 Agent 如何处理不确定性。

【例 7-4】 考虑例 7-2 所示的数据, 假定没有输入特征, 则所有样本的预测值是一样的。

在第一种表示方法中, 在训练数据上最小化绝对误差的预测值是 2, 相应的绝对误差是 10。而最小化平方和误差与最坏情况误差的预测值分别是 3.2 和 3.5。

在第二种表示方法中, 在训练数据上最小化绝对误差的预测值是 $Y_1=0$ 。最小化平方和误差的预测值为 $Y_1=0.4$ 、 $Y_2=0.1$ 、 $Y_3=0$ 、 $Y_4=0$ 、 $Y_5=0$ 、 $Y_6=0.4$, 这也是最小化熵和最大化训练数据的似然性的预测结果。最小化最坏情况误差的预测值是 $Y_1=Y_2=Y_6=0.5$ 、 $Y_3=Y_4=Y_5=0$ 。

293

因此, 采用哪种预测依赖于如何评估预测。

7.2.2 无输入特征的点估计

学习的最简单情形是数据没有输入特征且只有一个目标特征。对于许多学习算法来讲, 这是基准情形, 相当于忽略了所有的输入特征。在这种情形下, 学习算法对于所有的样本均给出单一目标特征值的预测。最小化误差的预测依赖于采用哪种误差。

假定 E 是样本集合, Y 是一个数值特征。Agent 能做的事就是对所有样本给出一个单一点估计值。注意, 虽然 Agent 进行随机预测是可能的, 但这样做并非更好, 参见习题 7.2。

E 上的预测值为 v 的平方和误差是

$$\sum_{e \in E} (\text{val}(e, Y) - v)^2$$

E 上的预测值为 v 的绝对误差是

$$\sum_{e \in E} |\text{val}(e, Y) - v|$$

E 上的预测值为 v 的最坏情况误差是

$$\max_{e \in E} |val(e, Y) - v|$$

命题 7.1 假定 V 是一个多值集合, 集合的元素为 $val(e, Y)$, $e \in E$.

(a) E 上的最小化平方和误差的预测是 V 的平均值。

(b) E 上的最小化绝对误差的预测是 V 的中位值。特别的, 对于最小化误差的数字 v , V 中小于 v 的数值个数与大于 v 的数值个数相等。

(c) E 上的最小化最坏情况误差的预测是 $(max + min)/2$, 其中 max 和 min 分别是 V 的最大值和最小值。

证明: 详细的证明留作练习, 这里只给出基本的证明思路。

(a) 将平方和误差公式对于 v 求导数并设为 0。这属于初等微积分的内容。为了确保找到最小值, 不仅需要检查导数为 0 的点, 也需要检查区间的端点。

(b) 绝对误差是 v 的分段线性函数。不属于 V 的值的斜率依赖于大于该值的元素个数与小于该值的元素个数的差: 如果大于 v 的元素个数与小于 v 的元素个数相同, 则 v 是使得误差最小值化的一个预测。

(c) 这个预测的误差是 $(max - min)/2$, 增加或减少预测值都会加大误差。 ■

当目标特征的取值范围为 $\{0, 1\}$ 时, 训练样本可划分为: 值为 0 的样本个数 n_0 和值为 1 的样本个数 n_1 。每一个新样本的预测值都是一样的, 记为 p 。

最优预测 p 依赖于优化的标准。训练数据上优化标准的值可以被解析地计算出来, 结果如图 7-3 所示。

预测的度量	训练数据上预测值为 p 的度量	训练数据上的最优预测值
绝对误差	$n_0 p + n_1 (1 - p)$	$median(n_0, n_1)$
平方和误差	$n_0 p^2 + n_1 (1 - p)^2$	$\frac{n_1}{n_0 + n_1}$
最坏情况误差	$\begin{cases} p & \text{如果 } n_1 = 0 \\ 1 - p & \text{如果 } n_0 = 0 \\ \max(p, 1 - p) & \text{其他} \end{cases}$	$\begin{cases} 0 & \text{如果 } n_1 = 0 \\ 1 & \text{如果 } n_0 = 0 \\ 0.5 & \text{其他} \end{cases}$
似然性	$p^{n_1} (1 - p)^{n_0}$	$\frac{n_1}{n_0 + n_1}$
熵	$-n_1 \log p - n_0 \log (1 - p)$	$\frac{n_1}{n_0 + n_1}$

图 7-3 当训练数据由 n_0 个目标特征值为 0 的样本和 n_1 个目标特征值为 1 的样本构成(没有输入特征)。当 $n_0 > n_1$ 时, $median(n_0, n_1)$ 取值 0; 当 $n_0 < n_1$ 时, $median(n_0, n_1)$ 取值 1; 当 $n_0 = n_1$ 时, $median(n_0, n_1)$ 可以取在 $[0, 1]$ 范围内的任意值

注意, 优化绝对误差意味着预测取中位值; 由于误差是 p 的线性函数, 所以这并不令人惊讶。

基于其他优化标准的预测是计算经验频率(empirical frequency): 训练数据中取值为 1 的样本个数, 即 $\frac{n_1}{n_0 + n_1}$ 。这可以看成是一种概率(probability)的预测。经验频率也常称为最大似然估计(maximum-likelihood estimate)。

上述分析没有指定测试数据上的最优预测值。我们不能指望训练数据上的经验频率就是测试数据上的最优预测值(该预测值最大化似然性或最小化熵)。如果 $n_0 = 0$ 或 $n_1 = 0$, 则所有训练数据都归于同一个类别。然而, 只要有一个测试样本不是这样分类的, 则似然性将为 0(最小的可能值), 熵将为无限大。详见习题 7.1。

7.2.3 概率学习

对于许多用于预测的度量来说,训练数据上的最优预测是经验频率。因此,进行点估计可以解释为学习一个概率。然而,经验频率通常并不是新样例概率的一个好的估计,这是因为 Agent 没有观察到变量的值并不意味着该值应该赋予概率 0。概率为 0 说明该值是不可能的。

通常我们获得数据并非没有任何先验知识。大量的关于诸如符号的意义、类似样例的经验等领域的先验知识可用来提高预测的效果。

考虑 0 概率问题和先验知识的一般方法是利用伪计数(pseudocount)或先验计数(prior count)在训练数据上加上某个值。

假定有一个二元特征 Y , Agent 观察到有 n_0 个 $Y=0$ 的样例和 n_1 个 $Y=1$ 的样例。Agent 可以对 $Y=0$ 和 $Y=1$ 的情况分别使用伪计数 $c_0 \geq 0$ 和 $c_1 \geq 0$, 此时概率可估计为

$$P(Y=1) = (n_1 + c_1) / (n_0 + c_0 + n_1 + c_1)$$

这个概率同时考虑了数据和先验知识。该公式在具有参数先验知识的条件下得到了验证(见 7.8 节)。选择伪计数是设计学习器的一个部分。

更一般的,假定 Y 的取值范围是 $\{y_1, \dots, y_k\}$ 。Agent 对每个 y_i 设定伪计数 c_i , 它们在 Agent 观察到数据之前选定。如果 Agent 此时观察到一些训练样本, 且 n_i 是 $Y=y_i$ 的样本个数, 则

$$P(Y=y_i) = \frac{c_i + n_i}{\sum_j c_j + n_j}$$

为了确定伪计数,考虑这样一个问题:“已经观察到一个 $Y=y_i$ 的样例与没有观察到这样的样例相比, Agent 信任 y_i 的程度提升了多少?”。如果没有观察到 $Y=y_i$ 的样例, Agent 是不可能信任 y_i 的, 因此 c_i 应该为 0; 否则, 上述问题的答案应该是比例值 $(1+c_i):c_i$ 。由此可以看出:若伪计数取 1, 观察过一次的值出现的可能性是从没观察过的值的两倍;若伪计数取 10, 与从没观察过的值相比, 观察过一次的值出现的可能性增加了 10%;若伪计数取 0.1, 观察过一次的值出现的可能性是从没观察过的值的 11 倍。如果没有理由更偏好选择 Y 中的某个值, 所有的伪计数 c_i 应取同样的值。

如果没有先验知识, Laplace[1812]建议 c_i 取 1, 其合理性的说明参见 7.8 节。然而, 我们也会碰到这种取法并不恰当的实例。

相同的思想可用于学习条件概率分布(conditional probability distribution)。为了估计变量 Y 条件依赖于变量 X 的条件概率分布 $P(Y|X)$, Agent 可以为每一对 Y 和 X 的值维护一个计数。假定一个非负的值 c_{ij} 将被用于 $Y=y_i \wedge X=x_j$ 的伪计数, 且观察到的满足 $Y=y_i \wedge X=x_j$ 的样例个数是 n_{ij} , 则

$$P(Y=y_i | X=x_j) = \frac{c_{ij} + n_{ij}}{\sum_l c_{lj} + n_{lj}}$$

但当分母很小的时候, 上述公式就不能很好地解决问题。当 X 中的某些值非常稀少的时候就会出现分母很小的情形, 例如, 当 X 具有一定的结构(例如由其他变量构成), 此时在训练数据中经常出现 X 的某些赋值非常稀少, 甚至从不出现的情形。在这种情况下, 学习器必须采用将在本章后续介绍的其他方法。

从专家那里获得概率

伪计数的使用为我们提供了一种结合专家观点(expert opinion)和数据的方法。Agent 通常没有质量很好的数据, 但可能有咨询多位专家的条件, 这些专家拥有不同层次的专业

知识并可给出不同的概率。

从专家那里获得概率面临一系列的问题：

- 专家不愿意提供一个准确的概率值(该值不能被进一步精化)。
- 概率估计的不确定性的表示。
- 组合不同专家给定的概率值。
- 组合专家观点与实际数据。

与期望专家直接给出概率不同，专家可以提供某种计数值。不是给定一个诸如 0.667 那样的实数作为 A 的概率，专家可以给出一对诸如 $\langle n, m \rangle$ 那样的数值，它可以解释为专家从 m 次实验中观察到 n 次 A 的出现。本质上，专家不仅给出了一个概率，而且给定了数据集(数据集是专家观点形成的基础)大小的一个估计。

通过为系统增加伪计数组件，我们可以将不同专家的计数组合在一起。鉴于比率可以反映概率的大小，其绝对值可以用于表达不同级别的可信度： $\langle 2, 3 \rangle$ 表达了问题将由数据或专家估计所支配的一种特别低的可信度； $\langle 20, 30 \rangle$ 表达了更高的可信度——少量样本难以改变现状，但是几十个样本却可以。以此类推，几百个样本对先验计数对 $\langle 2000, 3000 \rangle$ 产生不了什么影响；然而，在面临成千上万个数据点的时候，这样的计数对对结果概率影响甚微。

297

7.3 有监督学习的基本模型

学习最终获得的模型是从输入特征到目标特征的一种函数表示。大多数有监督学习方法接收输入特征、目标特征和训练数据，返回能用作未来预测的模型。许多学习方法的区别在于采用什么方式表示函数。本节我们首先介绍一些基本的学习模型，它们可组合成更复杂的模型。7.4 节介绍由基本模型组成的复杂学习模型。

7.3.1 决策树学习

决策树是对样例进行分类的一种简单表示形式。在有监督分类学习中，决策树学习是最成功的技术之一。在本节中，假定所有的特征都取有限的离散值，且仅有一个称为类别(classification)的目标特征。类别的每一个元素称为类(class)。

决策树(decision tree)或分类树(classification tree)是一棵树，其中，每一个内部(非叶子)节点都用一个输入特征进行标记；从内部节点引出的每条弧分别用该特征的一个可能值进行标记；树的每个叶子节点用类或类上的概率分布进行标记。

当对一个样例进行分类时，从树的根节点开始从上往下进行过滤(或匹配)：对于树上遇到的每个特征，沿着该特征在样例上的值所对应的弧往下走。当到达一个叶子节点时，返回该叶子节点所对应的类。

【例 7-5】 图 7-4 显示了两棵可能的决策树，它们基于图 7-1 所示的样例。两棵决策树都可根据用户的行为对样例进行分类。当使用左边的决策树时，首先确定特征 $Length$ 的值，如果是 $long$ ，则预测值为 $skips$ ；否则检查特征 $Thread$ 的值，如果是 new ，则预测值为 $reads$ ；否则检查特征 $Author$ 的值，仅

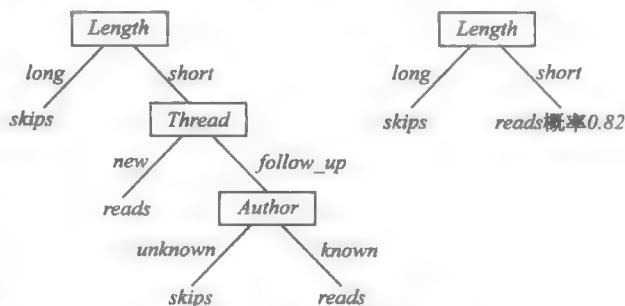


图 7-4 两棵决策树

当为 *known* 的时候才预测为 *reads*。这棵决策树能够正确分类如图 7-1 所示的所有样例。

右边的决策树当特征 *Length* 的值是 *short* 的时候进行概率预测。在这种情况下, 预测为 *reads* 和 *skips* 的概率分别是 0.82 和 0.18。

一棵确定的决策树(其中所有的叶子节点均代表类)可以被映射为一个规则的集合, 每一个叶子节点对应一条规则。如果从根节点到某个叶子节点的路径所包含的条件均为真, 则样例具有该叶子节点所代表的类。

【例 7-6】图 7-4 左边所示的决策树可用下述规则进行表示:

```
skips ← long
reads ← short ∧ new
reads ← short ∧ followUp ∧ known
skips ← short ∧ followUp ∧ unknown
```

如果把规则的否定看成是失败的, 则可以忽略结果是 *skips* 或 *reads* 的规则, 其他规则可以从反面推理得到。

使用决策树作为目标的表示形式存在如下一些问题:

- 给定一些训练样本, 应该生成什么样的决策树? 由于决策树能够表示输入特征的任意函数, 可以在偏好选择特定决策树的时候考虑学习的偏置问题。一个建议是选择与数据匹配的最小的决策树, 这意味着树的深度最浅或树的节点最少。哪一棵决策树是还未观察到的数据的最好预测者是一个经验方面的问题。
- Agent 如何处理建立一棵决策树的问题? 一种方法是在决策树空间中搜索适合数据的最小决策树。不幸的是, 决策树空间是非常巨大的(见习题 7.7)。一种实用的解决办法是在决策树空间中进行局部搜索, 目标是最小化误差。这就是下述所描述的算法的思想。

搜索一棵好的决策树

可以采用自顶向下的方式增量地构建一棵决策树, 它通过递归地选择一个特征进行分裂, 并依据该特征的值划分训练样本。如图 7-5 所示, 过程 *DecisionTreeLearner* 对二元属性(特征)学习一棵决策树。该过程并未考虑关于何时停止和选择什么特征进行分裂的问题。过程 *DecisionTreeClassify* 接收由 *DecisionTreeLearner* 产生的决策树和对新样本进行预测。

算法 *DecisionTreeLearner* 采用自顶向下的方式构建一棵决策树, 具体过程如下: 算法的输入是输入特征的集合、目标特征

```
1: procedure DecisionTreeLearner(X, Y, E)
2:   Inputs
3:     X: 输入特征的集合,  $X = \{X_1, \dots, X_n\}$ 
4:     Y: 目标特征
5:     E: 训练样本集
6:   Output
7:     决策树
8:   if 满足停止条件 then
9:     return pointEstimate(Y, E)
10:  else
11:    选择特征  $X_i \in X$ , 值域为  $\{v_1, v_2\}$ 
12:    令  $E_1 = \{e \in E: \text{val}(e, X_i) = v_1\}$ 
13:    令  $T_1 = \text{DecisionTreeLearner}(X \setminus \{X_i\}, Y, E_1)$ 
14:    令  $E_2 = \{e \in E: \text{val}(e, X_i) = v_2\}$ 
15:    令  $T_2 = \text{DecisionTreeLearner}(X \setminus \{X_i\}, Y, E_2)$ 
16:    return  $\langle X_i = v_1, T_1, T_2 \rangle$ 
17:
18: procedure DecisionTreeClassify(e, X, Y, DT)
19:   Inputs
20:     X: 输入特征的集合,  $X = \{X_1, \dots, X_n\}$ 
21:     Y: 目标特征
22:     e: 欲分类的样本
23:     DT: 决策树
24:   Output
25:     样例 e 的 Y 值预测
26:   Local
27:     S 是 DT 的子树
28:      $S_i = DT$ 
29:     while S 是一个内部节点且具有形式  $\langle X_i = v, T_1, T_2 \rangle$  do
30:       if  $\text{val}(e, X_i) = v$  then
31:          $S_i = T_1$ 
32:       else
33:          $S_i = T_2$ 
34:     return S
```

图 7-5 对于二元特征的决策树学习与分类

300

和样本集。学习器首先测试是否满足停止条件，如果满足就返回 Y 的一个点估计，这个点估计要么是 Y 的一个值，要么是 Y 取值的一个概率分布。如果不满足停止条件，学习器就选择一个特征 X_i 进行分裂，对于该特征的每一个值 v ，学习器为 $X_i = v$ 的所有样本递归地构建一棵子树。最终得到的树用 if-then-else 这种三元组的形式进行表示。

【例 7-7】 考虑在如图 7-1 所示的分类数据上执行 *DecisionTreeLearner* 过程。初始调用为

```
decisionTreeLearner([Author, Thread, Length, WhereRead], UserAction, [e1, e2, ..., e18])
```

假定停止条件不为真并选择特征 *Length* 作分裂，则调用

```
decisionTreeLearner([WhereRead, Thread, Author], UserAction, [e1, e3, e4, e5, e9, e10, e12])
```

此时所有的样本在用户的行为上取得了一致(即 *UserAction* 取相同的值——译者注)，因此算法返回预测值 *skips*。第二步递归调用是

```
decisionTreeLearner([WhereRead, Thread, Author], UserAction,
    [e2, e6, e7, e8, e11, e13, e14, e15, e16, e17, e18])
```

此时并非所有的样本在用户的行为上取得了一致，因此算法再选择一个特征作分裂，假定选择的特征是 *Thread*。最后，递归过程返回 *Length* 为 *short* 的子树，例如

```
<Thread = new, reads, (Author = unknown, skips, reads)>
```

最终的结果是

```
<Length = long, skips, (Thread = new, reads, (Author = unknown, skips, reads))>
```

这是如图 7-4 所示的树的一种表示形式。

如图 7-5 所示的学习算法有三个问题没有说明：

- 算法停止的条件。这些条件包括：没有输入特征、所有样本具有相同的分类，或者是不存在能够提升树的分类能力的特征分裂。最后一种条件是最难进行测试的，如下所述。
- 叶子节点应返回什么。这是一个点估计问题，因为这个步骤忽略了所有的其他输入特征。预测值通常包括：最有可能的分类、中位数或平均值、类的概率分布(见习题 7.9)。
- 选择哪个特征进行分裂。目标是选择这样的特征，它能够获得最小的决策树。一般的方法是选择**目前看来是最优**的特征(即贪心算法——译者注)作分裂：如果学习器仅允许一次分裂，哪种分裂能获得最优的分类？如果采用平方和误差准则，则对于每一个特征确定基于该特征的分裂得到的树的误差。如果采用似然性或熵的准则，则目前最优的分裂是能够获得**最大信息增益**(information gain)的分裂。信息增益有时也在采用平方和误差准则的情形下获得应用。习题 7.10 研究了基于**Gini 指数**(Gini index)的方法。

301

【例 7-8】 考虑从如图 7-1 所示的数据上学习用户的行为，我们选取特征进行分裂，该特征具有最大信息增益或最小化熵或最大似然性。信息的定义可参见 6.1.5 节。

所有样本关于特征 *UserAction* 的信息量是 1.0，因为 *UserAction* = *reads* 和 *UserAction* = *skips* 的样本个数相同，均为 9 个。

分裂特征 *Author* 将样本分为 *Author* = *known* 的集合 $[e_1, e_4, e_5, e_6, e_9, e_{10}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}]$ 和 *Author* = *unknown* 的集合 $[e_2, e_3, e_7, e_8, e_{11}, e_{18}]$ ，每一个集合均被不同的用户行为所平分(即 *UserAction* = *reads* 和 *UserAction* = *skips* 的样本个数相同)，所以特征 *Author* 的信息增益为 0。在这种情形下，知道 *Author* 的值并不能提供关于用户行为的任何信息。

分裂特征 *Thread* 将样本分为 $[e_1, e_2, e_5, e_8, e_{10}, e_{12}, e_{14}, e_{15}, e_{17}, e_{18}]$ 和 $[e_3, e_4, e_6, e_7, e_9, e_{11}, e_{13}, e_{16}]$ 。第一个集合 (*Thread*=new) 包含 3 个 *UserAction*=skips 的样本和 7 个 *UserAction*=reads 的样本, 因此关于用户行为的信息量为

$$-0.3 \times \log_2 0.3 - 0.7 \times \log_2 0.7 = 0.881$$

信息增益则为 0.119。

类似的, *Thread*=old 的样本数根据用户的行为可划分为 6:2 的比例, 信息量是 0.811。因此期望的信息增益是 $1.0 - [(10/18) \times 0.881 + (8/18) \times 0.811] = 0.150$ 。

分裂特征 *Length* 将样本分为 $[e_1, e_3, e_4, e_6, e_9, e_{10}, e_{12}]$ 和 $[e_2, e_5, e_7, e_8, e_{11}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}]$ 。前者在 *UserAction* 上的值相同, 因而信息量为 0。用户行为将第二个集合划分为 6:2 的比例, 信息量是 0.684。因此, 期望的信息增益是 $1.0 - 11/18 \times 0.684 = 0.582$ 。这是所有特征中最高的信息增益, 所以选择特征 *Length* 作分裂。

关于选择哪个特征作分裂, 在测试之前所有的特征的信息量是一样的, 因而学习 Agent 可以选择这样的特征, 它在测试之后的信息需求最小。

如图 7-5 所示的算法假定每个特征仅有两种值。可以通过下述方法放松这种限制:

- 允许多路分裂。分裂多值变量导致值域中的每一个值均对应着一棵子树。与简单的用于二元特征的 if-then-else 形式相比, 这意味着决策树的表示变得更为复杂。这里有两个问题需要考虑。第一个是怎样处理没有训练样本的特征值。第二个是对于大多数的贪心分裂启发式规则(包括信息增益)来讲, 在具有更多值的变量上作分裂效果往往更好。原因是, 与具有较少值个数的特征相比, 它能产生更多的子树以便与数据匹配(见习题 7.8)。然而, 在具有较少值个数的特征上作分裂能使得表示更为简洁。
- 将值域划分为两个不相交的子集。当值域是全序的(例如是实数集的子集), 可以根据某个阈值将值域划分为小于该阈值和大于该阈值的两个子集。例如, 根据 *X* 值域中的某个值 v , 可以得到 $X < v$ 的子树和 $X \geq v$ 的子树。 v 的目前最优值的选择可以这样来实现: 首先根据 *X* 值的大小进行排序, 然后考察依据各个值划分的效果。当值域不存在一个自然的排序关系时, 可以在值域的子集上进行任意划分。在这种情形下, 目前最优的划分可以通过数据基于类别概率的排序来找到。

如果数据中存在噪声, 上述算法的一个主要问题是过拟合(overfitting)。当算法试图匹配出现在训练数据中的某些特性(还未出现的样本并不具有这样的特性)时产生了过拟合。换句话说, 存在于训练数据中的随机关系并不反映数据集的整体关系时会产生过拟合。7.5 节会讨论检测过拟合的方法。存在两种解决决策树中过拟合问题的方法:

- 限制分裂操作, 即仅当分裂特征确实有用时才进行分裂。
- 允许不受限制的分裂, 但对最终的决策树(该决策树包含一些无根据的特性)进行剪枝。

在实践中, 第二种方法的效果更好。一个原因是存在这样的情况: 两个特征一起可以预测得很好, 但单独的一个特征并不十分有效。这可以从下面的例子中看出。

【例 7-9】 假定我们的目标是预测匹配便士博弈的赢或输。输入特征有三个: *A*、*B* 和 *C*, *A* 代表第一枚硬币的正面或反面, *B* 代表第二枚硬币的正面或反面, *C* 代表是否有喝彩。当两枚硬币均为正面(或反面)时, 目标特征 *W* 为真(即为赢)。此外, 假定有喝彩也说明赢了。这个例子设计得颇为巧妙, 因为 *A* 或 *B* 单独均无法提供有关 *W* 取值的任何信息, 可是 *A* 和 *B* 一起却可以准确地预测 *W* 的值。一种可能的分裂方法是首先选择特征 *C*

进行分裂,因为它提供最大的信息量。如果所有的 Agent 均被告知选择 C,这样做比选择 A 或 B 更有效。然而,如果最终选择了 A 和 B 进行分裂,则在 C 上的分裂是不必要的。剪枝操作可能删除 C,即使它是有用的;而算法提前结束将会选择 C 上的分裂。

303

关于如何在模型复杂性与数据匹配之间进行折中的问题将在 7.5 节进行讨论。

7.3.2 线性回归与分类

线性函数是许多学习算法的基础。本节首先讨论回归——从训练数据预测实值函数的问题,然后讨论分类问题(目标特征取离散值)。

线性回归(linear regression)指的是寻找一个线性函数去拟合训练样本集的输入-输出对,其中输入特征和输出特征均为数值型。

假定输入特征是 X_1, \dots, X_n , 则**线性函数**(linear function)的形式为

$$f^{\bar{w}}(X_1, \dots, X_n) = w_0 + w_1 \times X_1 + \dots + w_n \times X_n$$

其中 $\bar{w} = \langle w_0, w_1, \dots, w_n \rangle$ 是权值数组。为了使得 w_0 不显得那么特殊,可以引入一个新的特征 X_0 , 它的值总是 1。

我们将为每一个目标特征独立地学习一个函数,这里考虑仅有一个目标特征 Y 的情形。假定存在一个样本集 E , 其中每个样本 $e \in E$ 在特征 X_i 上的值为 $val(e, X_i)$, 在目标特征上的观察值为 $val(e, Y)$, 则目标特征的预测值是

$$pval^{\bar{w}}(e, Y) = w_0 + w_1 \times val(e, X_1) + \dots + w_n \times val(e, X_n) = \sum_{i=0}^n w_i \times val(e, X_i)$$

其中我们显式地表达预测是依赖于权值的,且 $val(e, X_0) = 1$ 。

在样本集 E 上目标特征 Y 的平方和误差是

$$Error_E(\bar{w}) = \sum_{e \in E} (val(e, Y) - pval^{\bar{w}}(e, Y))^2 = \sum_{e \in E} (val(e, Y) - \sum_{i=0}^n w_i \times val(e, X_i))^2 \quad (7.1)$$

在这种线性情形下,可以解析地计算出最小化误差的权值(见习题 7.5)。更一般的方法(适用于更广泛的函数类型)是迭代地计算权值。

梯度下降就是一种寻找函数最小值的迭代方法。它首先设定初始的权值;然后在每一次迭代中都减少权值(减少的量与其偏导数成比例):

$$w_i := w_i - \eta \times \frac{\partial Error_E(\bar{w})}{\partial w_i}$$

304

其中梯度下降的步长 η 称为**学习率**(learning rate)。学习率、特征以及数据都属于学习算法的输入部分。偏导数指定权值的改变如何影响误差的改变。

考虑最小化平方和误差,它是样本集上的误差之和。由于和的偏导数等于偏导数之和,所以可以单独考虑每一个样本及其对权值改变的影响。样本 e 的误差对于权值 w_i 的偏导数是 $-2 \times [val(e, Y) - pval^{\bar{w}}(e, Y)] \times val(e, X_i)$ 。对于每一个样本 e , 设 $\delta = val(e, Y) - pval^{\bar{w}}(e, Y)$ 。由此可得样本 e 更新权值 w_i 的公式:

$$w_i := w_i + \eta \times \delta \times val(e, X_i) \quad (7.2)$$

上式中我们忽略了常数 2,因为它可以被常数 η 合并。

图 7-6 给出了一个名为 $LinearLearner(X, Y, E, \eta)$ 的算法,它学习一个最小化平方和误差的线性函数。注意在第 17 行,对于所有的 e , $val(e, X_0) = 1$ 。算法的终止条件通常包括:达到给定的迭代次数、误差足够小或值的改变足够小。


```

1: procedure LinearLearner( $X, Y, E, \eta$ )
2:   Inputs
3:      $X$ : 输入特征的集合,  $X = \{X_1, \dots, X_n\}$ 
4:      $Y$ : 目标特征
5:      $E$ : 用于学习的样本集
6:      $\eta$ : 学习率
7:   Output
8:     参数  $w_0, \dots, w_n$ 
9:   Local
10:     $w_0, \dots, w_n$ : 实数
11:     $pval^w(e, Y) = w_0 + w_1 \times val(e, X_1) + \dots + w_n \times val(e, X_n)$ 
12:    随机初始化  $w_0, \dots, w_n$ 
13:    repeat
14:      for  $E$  中的每一个样本  $e$  do
15:         $\delta_i = val(e, Y) - pval^w(e, Y)$ 
16:        for 每一个  $i \in [0, n]$  do
17:           $w_i = w_i + \eta \times \delta \times val(e, X_i)$ 
18:      until 终止
19:    return  $w_0, \dots, w_n$ 

```

图 7-6 采用梯度下降法学习一个线性函数

上述算法有时也称为**增量梯度下降**(incremental gradient descent), 因为它通过每一个样本的迭代来改变权值。另一种可行的方法是: 首先在每次循环迭代中保存权值, 然后利用这些权值计算函数值, 最后在所有的样本都处理完毕后更新权值。这个过程计算误差函数的真正导数, 但是它更为复杂且效果通常没有增量梯度下降法好。

上述算法同样适用于其他误差函数。对于绝对误差来讲, 它在 0 点并非真正可微, 但在该点的导数可定义为 0, 因为此时误差已经达到最小, 参数也已经无须改变。见习题 7.12。

扁线性函数

对于分类任务来讲, 线性函数的效果并不好。当仅存在二值(即 0 和 1)时, 学习器应该决不会给出大于 1 或小于 0 的预测值。可是为了更好地匹配其他样本, 线性函数可能为某个样本给出值为 3 的预测。

我们首先考虑二分类, 其目标变量的值域是 $\{0, 1\}$ 。如果存在多重二元目标变量, 它们可以单独地被学习。

对于分类问题, 我们经常使用一个**扁线性函数**(squashed linear function):

$$f^w(X_1, \dots, X_n) = f(w_0 + w_1 \times X_1 + \dots + w_n \times X_n)$$

f 也称为**激活函数**(activation function), 它将实数映射到 $[0, 1]$ 。利用扁线性函数预测目标特征的值, 可以得到样本 e 在目标特征 Y 上的预测值:

$$pval^w(e, Y) = f(w_0 + w_1 \times val(e, X_1) + \dots + w_n \times val(e, X_n))$$

一种简单的激活函数是**阶跃函数**(step function) $f(x)$, 定义为

$$f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

阶跃函数是**感知器**(perceptron)[Rosenblatt, 1958]的基础, 感知器是最早提出的学习算法之一。为阶跃函数应用梯度下降法是困难的, 因为梯度下降需要导数而阶跃函数是不可微的。

如果激活函数是可微的, 则可以利用梯度下降法更新权值。平方和误差是

$$Error_E(\bar{w}) = \sum_{e \in E} (val(e, Y) - f(\sum_i w_i \times val(e, X_i)))^2$$

对于样本 e 而言, 该误差关于权值 w_i 的偏导数是

$$\frac{\partial \text{Error}_e(\bar{w})}{\partial w_i} = -2 \times \delta \times f' \left(\sum_j w_j \times \text{val}(e, X_j) \right) \times \text{val}(e, X_i)$$

其中 $\delta = \text{val}(e, Y) - p\text{val}(\bar{w})(e, Y)$ 。由此可得, 样本 e 更新权值 w_i 的公式如下:

306

$$w_i := w_i + \eta \times \delta \times f' \left(\sum_j w_j \times \text{val}(e, X_j) \right) \times \text{val}(e, X_i)$$

一种典型的可微的激活函数是 sigmoid 或 logistic 函数:

$$f(x) = \frac{1}{1 + e^{-x}}$$

该函数的图像如图 7-7 所示, 它将实值线挤压到区间 $(0, 1)$ 。logistic 函数适用于分类问题, 因为我们不想给出值大于 1 或小于 0 的预测。它同时也是可微的且导数仅由输出值决定, 即 $f'(x) = f(x) \times (1 - f(x))$ 。

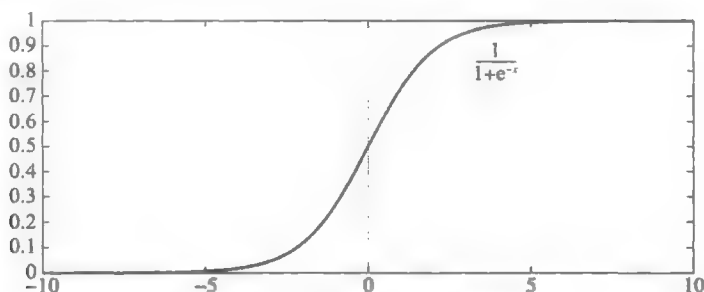


图 7-7 sigmoid 或 logistic 函数

可以将如图 7-6 所示的算法 *LinearLearner* 作一些改变, 以利用 sigmoid 函数。这可以通过将算法的第 17 行替换为下式来实现:

$$w_i := w_i + \eta \times \delta \times p\text{val}(\bar{w})(e, Y) \times [1 - p\text{val}(\bar{w})(e, Y)] \times \text{val}(e, X_i)$$

其中 $p\text{val}(\bar{w})(e, Y) = f(\sum_j w_j \times \text{val}(e, X_j))$ 是样本 e 的目标特征 Y 的预测值。

【例 7-10】 考虑学习一个扁线性函数, 目标是对如图 7-1 所示的数据进行分类。能正确实分类这些样本的一个函数是

$$\text{Reads} = f(-8 + 7 \times \text{Short} + 3 \times \text{New} + 3 \times \text{Known})$$

其中 f 是 sigmoid 函数。通过大约 3 000 次的梯度下降迭代(学习率 $\eta = 0.05$)可以找到类似上式的函数。根据这个函数, *Reads* 为真(与 0 相比, 预测值更接近 1), 当且仅当 *Short* 为真和 *New* 或 *Known* 为真。因此, 这个线性分类器学到的函数与决策树是一样的。上述工作的细节可参考 Alspace.org 网站上神经网络中的“mail reading”Java 小程序。

307

采用 sigmoid 函数作为激活函数的算法可以处理任意线性可分的分类问题。换句话说, 在任意样本集上的学习误差可以任意小, 当且仅当目标分类是线性可分的。一个分类是线性可分(linearly separable)的, 指的是存在一个超平面使得在该超平面的一侧类别为真, 另一侧为假。超平面定义为预测值, $f(\bar{w})(X_1, \dots, X_n) = f(w_0 + w_1 \times \text{val}(e, X_1) + \dots + w_n \times \text{val}(e, X_n))$, 是 0.5 的那个平面。对于 sigmoid 激活函数来讲, 当所得的权值 \bar{w} 满足 $w_0 + w_1 \times \text{val}(e, X_1) + \dots + w_n \times \text{val}(e, X_n) = 0$ 时得到该超平面。在该超平面的一侧, 预测值大于 0.5, 而在另一侧则小于 0.5。

图 7-8 给出了 or 和 and 的线性分类器。短划线将正(真)类和负(假)类分开。一种简单的

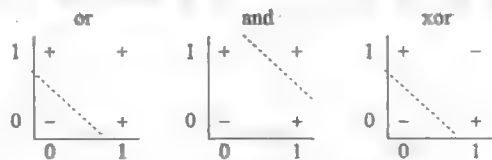


图 7-8 布尔函数的线性分类器

线性不可分函数是异或(xor)函数(见图的右边),此时不存在将正类样本和负类样本分开的直线。因此,线性分类器无法表示和学习异或函数。

通常先验地确定一个数据集是否线性可分是困难的。

【例 7-11】 考虑如图 7-9 所示的数据集,它用于预测某人是否喜欢去度假,这取决于下面这些因素:是否有文化积淀?是否必须坐飞机?目的地的天气很热吗?有音乐吗?是否有自然景观?在该数据集中,1 表示真,0 表示假。线性分类器要求用数值表示的形式。

经过 10 000 次的梯度下降迭代(学习率取 0.05)之后,最优的预测函数为(保留一位小数):

$$\text{Likes} = f(2.3 \times \text{Culture} + 0.01 \times \text{Fly} - 9.1 \times \text{Hot} - 4.5 \times \text{Music} + 6.8 \times \text{Nature} + 0.01)$$

除了最后和倒数第三个样本,该函数近似地预测了训练集中所有样本的目标值。最后和倒数第三个样本的预测值约为 0.5。这个函数对于不同的初始化值显得十分稳定。增加迭代的次数使得预测其他样本更为准确。

当目标变量的取值个数大于 2(即多分类问题),可以利用指示变量将多分类问题转化为二分类问题。这些二分类问题可以单独地被学习。然而,必须将这些单独的分类器组合起来预测目标变量的值。由于对于每一个样本来说必须恰好有一个值为真,所以学习器不能将多个值预测为真或者没有一个值为真。预测值为概率分布的分类器可以将多个单独的预测加以标准化。必须给出确定性预测的学习器可以采用上述模型。

Culture	Fly	Hot	Music	Nature	Likes
0	0	1	0	0	0
0	1	1	0	0	0
1	1	1	1	1	0
0	1	1	1	1	0
0	1	1	0	1	0
1	0	0	1	1	1
0	0	0	0	0	0
0	0	0	1	1	1
1	1	1	0	0	0
1	1	0	1	1	1
1	1	0	0	0	1
1	0	1	0	1	1
0	0	0	1	0	0
1	0	1	1	0	0
1	1	1	1	0	0
1	0	0	1	0	0
1	1	1	0	1	0
0	0	0	0	1	1
0	1	0	0	0	1

图 7-9 关于某人喜欢去哪里度假的训练数据

308

7.3.3 贝叶斯分类器

贝叶斯分类器(Bayesian classifier)基于这样的思想:一个(自然)类的作用是预测该类成员的特征值。样本组成类是因为它们具有相同的特征值,这样的类往往称为**自然类**(natural kind)。在本节中,目标特征对应于取离散值的类别,且该类别并不一定是二分类的。

如果 Agent 知道了类别,它就能预测其他特征的值——这是贝叶斯分类器的基本思想。如果 Agent 不知道类别,那么在给定一些特征值的前提下,贝叶斯规则可用于预测类别。在贝叶斯分类器中,学习 Agent 首先建立一个特征的概率模型,然后利用该模型预测新样本的类别。

潜在变量指的是未被观察到的概率变量。贝叶斯分类器是一种概率模型,其中的类别是与观察到的变量概率相关的潜在变量。分类因而转化成概率模型中的推理。

最简单的形式是**朴素贝叶斯分类器**,它基于独立性假设:在给定类别的前提下,输入特征之间是条件独立的。(条件独立的定义见 6.2 节。)

可用一个特殊的信念网络来表示朴素贝叶斯分类器的独立性。在该网络中,特征作为节点,目标变量(类别)没有父亲节点,类别是每一个输入特征的唯一父亲。这样的信念网络需要目标特征 Y 的概率分布 $P(Y)$ 和每一个输入特征 X_i 的概率分布 $P(X_i|Y)$ 。对于每一个样本,可依据观察到的输入特征值和类别信息计算预测值。

309

给定具有输入 $X_1 = v_1, \dots, X_k = v_k$ 的样本, 可利用贝叶斯规则计算该样本类别 Y 的后验概率分布:

$$\begin{aligned} P(Y | X_1 = v_1, \dots, X_k = v_k) \\ &= \frac{P(X_1 = v_1, \dots, X_k = v_k | Y) \times P(Y)}{P(X_1 = v_1, \dots, X_k = v_k)} \\ &= \frac{P(X_1 = v_1 | Y) \times \dots \times P(X_k = v_k | Y) \times P(Y)}{\sum_Y P(X_1 = v_1 | Y) \times \dots \times P(X_k = v_k | Y) \times P(Y)} \end{aligned}$$

其中分母是一个归一化常数以确保概率和为 1。由于分母不依赖于具体的类, 所以并不需要确定最可能的类。

为了学习一个分类器, 如 7.2.3 节所述, 可以从数据中计算得到每个输入特征的概率分布 $P(Y)$ 和 $P(X_i | Y)$ 。最简单的方法是将训练数据中的经验频率作为概率(即利用占训练数据的比例作为概率)。但是, 当存在 0 概率的时候这并非是一个好方法, 如下所述。

【例 7-12】 给定如图 7-1 所示的数据, 假定 Agent 想预测用户的行为。在这个例子中, 用户的行为就是类别。这里的朴素贝叶斯分类器对应于如图 7-10 所示的信念网络。训练样本用于确定信念网络所需的概率。

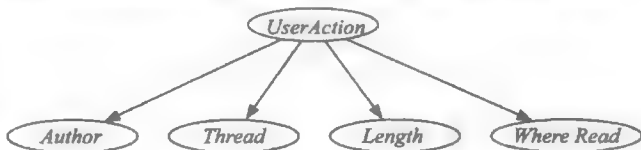


图 7-10 对应于一个朴素贝叶斯分类器的信念网

假定 Agent 利用经验频率作为样本的概率。这些可从数据中导出的概率如下:

$$P(\text{UserAction} = \text{reads}) = \frac{9}{18} = 0.5$$

$$P(\text{Author} = \text{known} | \text{UserAction} = \text{reads}) = \frac{2}{3}$$

$$P(\text{Author} = \text{known} | \text{UserAction} = \text{skips}) = \frac{2}{3}$$

$$P(\text{Thread} = \text{new} | \text{UserAction} = \text{reads}) = \frac{7}{9}$$

$$P(\text{Thread} = \text{new} | \text{UserAction} = \text{skips}) = \frac{1}{3}$$

$$P(\text{Length} = \text{long} | \text{UserAction} = \text{reads}) = 0$$

$$P(\text{Length} = \text{long} | \text{UserAction} = \text{skips}) = \frac{7}{9}$$

$$P(\text{whereRead} = \text{home} | \text{UserAction} = \text{reads}) = \frac{4}{9}$$

$$P(\text{WhereRead} = \text{home} | \text{UserAction} = \text{skips}) = \frac{4}{9}$$

从这些概率可以看出, 特征 *Author* 和 *WhereRead* 没有预测能力, 因为它们的取值不影响用户将阅读文章的概率。本实例的剩余部分将忽略这些特征。

对于一个新样本(特征 *Author*、*Thread*、*Length* 和 *WhereRead* 的值分别是 *unknown*、*followUp*、*short* 和 *home*), 我们有

$$\begin{aligned} P(\text{UserAction} = \text{reads} | \text{Thread} = \text{followUp} \wedge \text{Length} = \text{short}) \\ &= P(\text{followUp} | \text{reads}) \times P(\text{short} | \text{reads}) \times P(\text{reads}) \times c \\ &= \frac{2}{9} \times 1 \times \frac{1}{2} \times c \\ &= \frac{1}{9} \times c \end{aligned}$$

$$\begin{aligned}
 P(\text{UserAction} = \text{skips} \mid \text{Thread} = \text{followUp} \wedge \text{Length} = \text{short}) \\
 &= P(\text{followUp} \mid \text{skips}) \times P(\text{short} \mid \text{skips}) \times P(\text{skips}) \times c \\
 &= \frac{2}{3} \times \frac{2}{9} \times \frac{1}{2} \times c \\
 &= \frac{2}{27} \times c
 \end{aligned}$$

其中 c 是归一化常数, 用以保证上述两者的和为 1。显然, c 必须为 $\frac{27}{5}$, 所以

$$P(\text{UserAction} = \text{reads} \mid \text{Thread} = \text{followUp} \wedge \text{Length} = \text{short}) = 0.6$$

[311]

这样的预测对于样本 e_{11} 无效, 即虽然 Thread 取 followUp 且 Length 取 short , 用户的行为却是 skips 。朴素贝叶斯分类器将数据归结为一些参数的求取。它预测用户将阅读文章的原因是: $\text{Length} = \text{short}$ 是将阅读文章的信号, $\text{Thread} = \text{followUp}$ 是将略过文章的信号, 前者的指示强度大于后者。

对于 Length 取 long 的新样本, 我们有 $P(\text{Length} = \text{long} \mid \text{UserAction} = \text{reads}) = 0$ 。因此, 无论其他特征取什么值, $\text{UserAction} = \text{reads}$ 的后验概率都等于 0。

0 概率的使用会导致一些不可预测的后果。首先, 一些特征变得具有预测能力: 仅仅知道一个特征的值就能排除一个类别。如果允许 0 概率, 组合一些观察值是不可能的, 见习题 7.13。使用贝叶斯分类器, 0 概率问题并非一定是必然的, 但将经验频率作为概率就会出现 0 概率问题。使用经验频率(同时避免 0 概率问题)的一个可行方法是引进伪计数。正如下面的例子所述, 学习器的设计者应该小心地选择伪计数。

【例 7-13】 考虑例 6-16 所示的帮助系统, 其中帮助 Agent 根据用户给定的关键词推断用户感兴趣的帮助页面。对于这样的一个系统, 我们如何学习其中的概率? 帮助 Agent 必须知道两种概率: 每个帮助页面(这些页面是用户想要的)的先验概率; 给定帮助页面的前提下, 每个关键词的概率。必须学习这些概率的原因是系统设计者预先并不知道用户会使用什么词语。当用户寻求帮助的时候, Agent 能从用户真正使用的词语中学得这些概率。

学习器必须知道 $P(H)$ 。为此, 它首先给每个 h_i 一个伪计数。可以为那些预先知道将更可能被使用的页面设定更大的伪计数。如果设计者并没有关于什么页面将更可能被使用的先验信念, 则 Agent 可以为所有页面设定相同的伪计数。至于使用什么样的计数, 设计者必须考虑这样的问题: 在 Agent 看过一个页面之后, 它有多大程度信任该页面正是所需要的? 见 7.2.3 节。如果设计者已经通过优化训练数据上的伪计数(或通过使用层次贝叶斯模型)建立了另外一个帮助系统, 学习这样的伪计数是可能的。给定伪计数和一些数据, 可以计算 $P(h_i)$, 方法是将所有页面的计数和除以与 h_i 相关的计数(经验计数加上伪计数)。

对于词 w_j 和页面 h_i , 帮助 Agent 需要两种计数, 即当 h_i 是合适的页面时, 使用词 w_j 的个数(记为 c_{ij}^+)和未使用词 w_j 的个数(记为 c_{ij}^-), 它们都不应是 0。我们期望从平均意义上讲 c_{ij}^- 大于 c_{ij}^+ , 原因是我们希望一个查询平均上只使用所有词语中的一小部分词。我们可能为出现在帮助页面 h_i 里的那些词和不出现在 h_i 里的那些词设定不同的计数, 以便系统从合乎情理的状态开始运行。

[312]

每当用户声称找到了他们感兴趣的帮助页面时, 都要更新该页面的计数和所有词语的条件计数。具体的, 如果用户宣称 h_i 是其感兴趣的页面, 则增大与 h_i 相关联的计数 c_{ij}^+ 和 c_{ij}^- 。

上述模型没有使用有关错误页面(即不感兴趣的页面——译者注)的信息。如果用户声称一个页面不是他感兴趣的, 这种信息需要在找到正确的页面(即感兴趣的页面——译者

注)之后才可能被使用。

构建这样一个帮助系统的最大挑战不在于学习本身,而在于获得有用的数据。特别的,用户可能并不知道他们是否找到了他们想要的页面,因而也就可能不知道何时停止寻找和给出学习系统所需要的反馈信息。一些用户可能对所有页面都不满意。事实上,令用户满意的页面可能一个也不存在,但这种信息并没有反馈给学习者。另一方面,一些用户可能指出他们已经找到了想要的页面,即使可能存在另一个更合适的页面。后面一种情形可能导致正确页面的计数减少,从而使得发现不了该页面。◀

虽然存在一些朴素贝叶斯分类器的结果不好的情形,但总的来说它是一种非常简单、易于实现、大多数情况下效果非常好的学习算法。对于一个新问题,尝试使用朴素贝叶斯分类器是一种不错的选择。

一般而言,朴素贝叶斯分类器在满足独立性假设(即类是其他特征的一个很好的预测器和给定类的条件下其他特征是独立的)的时候工作良好。通过自然演化形成的自然类应该满足独立性假设,它们有助于人类区别不同的对象。自然类通常与名词相关,例如狗类或椅子类。

7.4 组合模型

决策树、扁线性函数和贝叶斯分类器是其他许多有监督学习技术的基础。线性分类器的表示能力是非常受限的。虽然决策树能够表示任意的离散函数,但是许多简单函数的决策树表示十分复杂。贝叶斯分类器使用一个先验的建模假设,它在实际问题中可能失效。

增强线性函数的表示能力的一种方法是:增加一些输入特征,使得线性函数变为原始输入特征的某种非线性函数。增加这些新特征可能增加空间的维度,使得某些函数在低维空间不是线性的(或线性可分的),但在高维空间却是线性的。

313

【例 7-14】 异或函数, $x_1 \text{ xor } x_2$, 在三维空间 X_1 、 X_2 和 x_1x_2 上是线性可分的, 其中 x_1x_2 是一个新特征, 其取值规则是: 当 x_1 和 x_2 两者都为真时结果为真。为了可视化这个结果, 考虑图 7-8: 通过增加 x_1x_2 为第三维特征, 图中右上角的点将越出页面边界, 从而允许获得一个线性分离器(此时是一个平面)。◀

支持向量机(support vector machine, SVM)可用于分类, 它利用原始输入的函数作为线性函数的输入。这些用作输入的函数称为**核函数**(kernel function)。存在许多可能的核函数, 原始特征的积就是一个例子, 增加特征的积作为新的特征足以表示异或函数。然而, 增加空间的维度可能导致过拟合。支持向量机构建一个决策面(超平面), 用以将正类样本和负类样本在高维空间中隔开。**间隔**(margin)定义为从决策面到任意样本的最小距离。支持向量机寻找具有最大间隔的决策面。最靠近决策面的样本(即支持向量)支撑起这个决策面。特别的, 如果删除这些样本, 则决策面会发生变化。支持向量使得能够用比样本个数少很多的参数定义决策面, 从而避免了过拟合。有关支持向量机的更详细描述见本章末尾的参考文献。

神经网络允许扁线性函数的输入是带有需调整参数的扁线性函数。将多层扁线性函数作为预测目标变量的扁线性函数的输入, 使得神经网络能够表示更为复杂的函数。神经网络将在后面作进一步阐述。

另一种非线性的表示形式是**回归树**(regression tree), 它是决策树的一种, 只不过叶子节点是一个扁线性函数。回归树可以近似地表示分段线性函数。与回归树类似, 决策树的叶子甚至可以是神经网络或支持向量机。为了分类一个新样本, 先将它顺着树往下过

滤,然后使用叶子节点上的分类器分类该样本。

可以对朴素贝叶斯分类器进行扩展,允许存在一些特征是类别的父节点和子节点。在给定父节点的前提下,类别的概率可以表示成决策树(或扁线性函数,或神经网络)。类别的子节点不必是独立的,可以表示为树增强朴素贝叶斯网络(tree augmented naive Bayesian network),在该网络中,子节点除了类别节点,还允许一个其他父节点的存在(只要最终的结果图是无环的)。这种简单的模型解释了子节点之间的相互依赖性。一种可替代的方法是在类变量中引入结构。潜在树模型(latent tree model)将类变量分解成一系列的潜在变量,这些变量通过树结构连接在一起。每一个观察到的变量都是其中一个潜在变量的子节点。潜在变量允许对观察到的变量之间的依赖性进行建模。

314

另外一种方法是使用多个分类器:首先在数据上分别训练这些分类器,然后利用一些诸如投票或线性函数的机制将这些分类器组合在一起。这种技术就是所谓的集成学习。

7.4.1 神经网络

神经网络是一种流行的用于学习的目标表示形式。神经网络是受到大脑神经元的启发而提出的,但并没有真正模拟神经元。与人类大脑中存在的大约 10^{11} 个神经元相比,人工神经网络包含的神经元(称为单元)个数通常要少得多;此外,与生物学意义上的神经元相比,这些单元要简单得多。

人工神经网络是一个有趣的研究课题,原因如下:

- 作为神经系统科学的一个部分,为了理解真正的神经系统,研究人员模拟了一些诸如蠕虫之类的低等动物的神经系统。这些研究有望理解究竟是神经系统的哪些方面解释了这些动物的行为。
- 一些研究人员试图不仅自动化实现智力的功能(人工智能研究的内容),而且自动化构造大脑的结构。一个假设是,实现大脑功能的唯一途径是使用与大脑一样的结构。测试该假设的方法是,分别通过使用与不使用大脑的结构来尝试智能的实现。构造其他机器(一个例子是飞行器,它使用与鸟飞行相同的原理,但结构不同)的实验预示着该假设可能不成立。虽然如此,测试该假设是一个令人感兴趣的工作。
- 大脑启发一种与现存计算机不同的计算新方法。与现在的计算机不同,它具有更少的处理器和大量的本质上是惰性的存储器,“大脑”由大量的异构的分布式过程组成(这些过程并发执行,不存在主控制器)。我们不应认为目前的计算机是用于计算的唯一体系结构。
- 与其他学习算法(例如决策树)相比,神经网络提供了一种不同的简洁性度量作为学习的偏置。多层神经网络和决策树一样,能表示离散特征集的任意函数。但是,能用简单神经网络表示的函数不一定能用简单的决策树表示。神经网络与决策树提供了不同的学习偏置,在实践中究竟哪个更好是一个经验性的问题,可以通过在不同领域的测试来判定。

315

存在许多不同种类的神经网络,本书考虑其中的一种,即前馈神经网络(feed-forward neural network)。前馈神经网络可以看成级联的扁线性函数。输入数据进入隐藏单元层(可以有多层),最终进入输出层。每个隐藏单元层都是其输入的扁线性函数。

这种神经网络的输入可以是多个任意实数,输出的是一个实数。对于回归问题,通常输出单元提供的是其输入的线性函数;而对于分类问题,通常输出单元提供的是其输入的sigmoid函数(因为不可能有超出 $[0, 1]$ 范围的预测值)。对于隐藏单元层来说,输出是其

输入的线性函数是不可能的，因为线性函数的线性函数是一个线性函数。增加额外的层次并不能带来新的功能。因此，每一个隐层单元的输出是其输入的一个扁线性函数。

与神经网络相关的是所有线性函数的参数。可以同时调优这些参数，方法是通过最小化训练样本上的预测误差来实现。

【例 7-15】 图 7-11 给出了只有一个隐层的神经网络，用于处理如图 7-9 所示的分类数据。如例 7-11 所解释，这个数据集不是线性可分的。该数据集有 5 个布尔输入特征，分别对应于是否有文化积淀、是否必须坐飞机、目的地是否很热、是否有音乐、是否有自然景观 5 个因素；唯一的输出特征对应于人是否喜欢去度假。在该神经网络中，只有一个包含两个隐藏单元的隐层，其中隐藏单元没有先验的含义。这个网络表示了下列公式：

$$\begin{aligned} \text{goal}(e, \text{Likes}) &= f(w_0 + w_1 \times \text{val}(e, H1) + w_2 \times \text{val}(e, H2)) \\ \text{val}(e, H1) &= f(w_3 + w_4 \times \text{val}(e, \text{Culture}) + w_5 \times \text{val}(e, \text{Fly}) \\ &\quad + w_6 \times \text{val}(e, \text{Hot}) + w_7 \times \text{val}(e, \text{Music}) + w_8 \times \text{val}(e, \text{Nature})) \\ \text{val}(e, H2) &= f(w_9 + w_{10} \times \text{val}(e, \text{Culture}) + w_{11} \times \text{val}(e, \text{Fly}) \\ &\quad + w_{12} \times \text{val}(e, \text{Hot}) + w_{13} \times \text{val}(e, \text{Music}) + w_{14} \times \text{val}(e, \text{Nature})) \end{aligned}$$

其中 $f(x)$ 是一个激活函数。

对于这个例子来说，需要学习 15 个实数(参数)(w_0, \dots, w_{14})，因此假设空间是一个 15 维的实值空间。空间中的每一个点对应于一个函数，它预测每个样本的输出值。

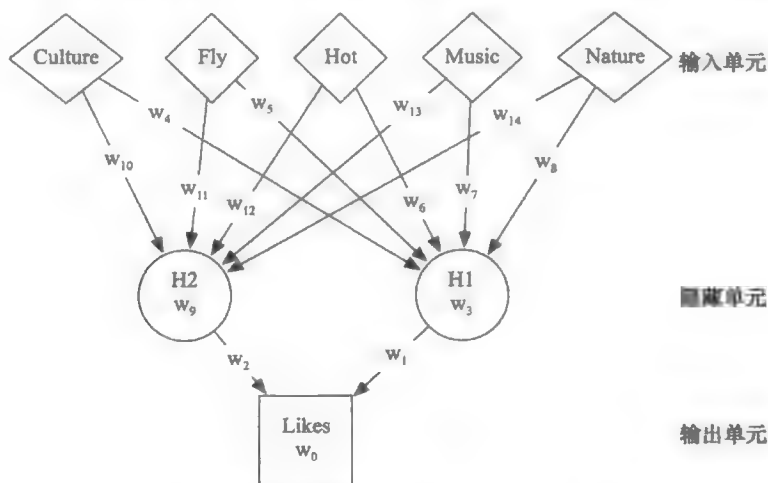


图 7-11 带有一个隐层的神经网络，其中 w_i 是权值。节点里面的权值不依赖于输入，其系数为 1。该神经网络的涵义见例 7-15

给定参数值和输入值，神经网络预测目标特征的值。神经网络的目标是：给定样本集，寻找最小化误差的参数值。如果存在 m 个参数，寻找具有最小误差的参数值涉及搜索一个 m 维的欧氏空间。

反向传播学习(back-propagation learning)是在参数空间进行梯度下降搜索以最小化平方和误差。

图 7-12 给出了带有一个隐层的神经网络的反向传播算法(增量梯度下降版本)。该算法假定有 n 个输入特征、 k 个输出特征和 n_k 个隐藏单元。 hw 和 ow 是用于保存权值的二维数组。0: n_k 是从 0 到 n_k (包括 n_k) 的下标，1: n_k 是从 1 到 n_k (包括 n_k) 的下标。该算法假定对于所有的 e , $\text{val}(e, X_0) = 1$ 。


```

1: procedure BackPropagationLearner( $X, Y, E, n_h, \eta$ )
2:   Inputs
3:    $X$ : 输入特征的集合,  $X = \{X_1, \dots, X_n\}$ 
4:    $Y$ : 目标特征的集合,  $Y = \{Y_1, \dots, Y_k\}$ 
5:    $E$ : 用于学习的样本集
6:    $n_h$ : 隐藏单元的个数
7:    $\eta$ : 学习率
8:   Output
9:   隐藏单元权值  $hw[0: n, 1: n_h]$ 
10:  输出单元权值  $ow[0: n_h, 1: k]$ 
11:  Local
12:   $hw[0: n, 1: n_h]$  隐藏单元权值
13:   $ow[0: n_h, 1: k]$  输出单元权值
14:   $hid[0: n_h]$  隐藏单元的值
15:   $hErr[1: n_h]$  隐藏单元的误差
16:   $out[1: k]$  输出单元的预测值
17:   $oErr[1: k]$  输出单元的误差
18:  随机初始化  $hw$  和  $ow$ 
19:   $hid[0] := 1$ 
20:  repeat
21:    for  $E$  中的每一个样本  $e$  do
22:      for 每一个  $h \in \{1, \dots, n_h\}$  do
23:         $hid[h] := \sum_{i=0}^n hw[i, h] \times val(e, X_i)$ 
24:      for 每一个  $h \in \{1, \dots, k\}$  do
25:         $out[o] := \sum_{h=0}^{n_h} hw[i, h] \times hid[h]$ 
26:         $oErr[o] := out[o] \times (1 - out[o]) \times (val(e, Y_o) - out[o])$ 
27:      for 每一个  $h \in \{0, \dots, n_h\}$  do
28:         $hErr[h] := hid[h] \times (1 - hid[h]) \times \sum_{o=0}^k ow[h, o] \times oErr[o]$ 
29:      for 每一个  $i \in \{0, \dots, n\}$  do
30:         $hw[i, h] := hw[i, h] + \eta \times hErr[h] \times val(e, X_i)$ 
31:      for 每一个  $o \in \{1, \dots, k\}$  do
32:         $ow[h, o] := ow[h, o] + \eta \times oErr[o] \times hid[h]$ 
33:    until 终止
34:  return  $w_0, \dots, w_n$ 

```

图 7-12 带有一个隐层的神经网络的反向传播学习算法

反向传播算法与如图 7-6 所示的线性学习器类似, 不同的是前者考虑了多层结构和激活函数。直观地, 对于每一个样本, 首先确定隐藏单元的值(第 23 行); 然后确定输出单元的值(第 25 行); 接着将误差通过网络反向传播(计算输出节点和隐藏节点上的误差分别见第 26 行和第 28 行); 最后基于误差的导数更新权值。

梯度下降搜索需要反复估算欲最小化的函数(这里指误差)及其导数。估算误差涉及在所有样本上的迭代计算。因此, 反向传播学习需要在所有样本上反复评估神经网络的性能。幸运的是, 我们采用了 logistic 函数——给定每个单元的输出值计算 logistic 函数的导数是很容易的。

【例 7-16】 如图 7-11 所示的带有一个隐层的神经网络(其中的隐层包含两个单元), 经过在如图 7-9 所示的数据的训练之后, 可以完美地匹配这些数据。

当学习率设为 $\eta = 0.05$ 时, 运行一次反向传播算法(经过 10 000 次迭代步骤), 最后学得的权值能准确地对训练数据进行预测:

$$H1 = f(-2.0 \times Culture - 4.43 \times Fly + 2.5 \times Hot + 2.4 \times Music - 6.1 \times Nature + 1.63)$$

$$H2 = f(-0.7 \times Culture + 3.0 \times Fly + 5.8 \times Hot + 2.0 \times Music - 1.7 \times Nature - 5.0)$$

$$Likes = f(-8.5 \times H1 - 8.8 \times H2 + 4.36)$$

神经网络的运用似乎对要求符号具有实际含义的**物理符号系统假设**(physical symbol system hypothesis)提出了挑战。对于神经网络的部分质疑是,虽然输入和输出单元符号的含义明确,但设计者并没有给出隐藏单元的实际含义。隐藏单元真正表示的是所学到的某些知识。在神经网络训练完毕之后,察看网络的内部细节以确定隐藏单元真正表示的是什么通常是可能的。有时候可以用语言简明地表达这种含义,但通常无法做到。然而,可以说计算机自有其内部的含义;通过考察样本是如何映射为隐藏单元的值可以解释其内部含义。

7.4.2 集成学习

在**集成学习**(ensemble learning)中,Agent 首先运行多个学习算法,然后组合这些算法的输出结果进行预测。用于组合的学习算法称为**基准算法**(base-level algorithm)。

集成学习的最简单情形是在数据的随机子集上训练基准算法,然后要么通过投票选择最可能的类别(对于确定性的预测而言),要么取基准算法预测的平均值作为最终的结果。例如,我们可以在随机样本集(占训练数据的50%)上训练多棵决策树,然后投票选择最可能的分类或取预测的平均值作为结果。决策树的输出甚至可以作为线性分类器的输入。线性分类器的权值可以通过学习获得。

这种方法在基准算法**不稳定**(unstable)的情形下工作良好:基准算法依据所选择的数据子集倾向于产生不同的表示形式。决策树和神经网络是不稳定的,但线性分类器的稳定性较好,因此采用线性分类器做集成效果可能较差。

在bagging方法中,若存在 m 个训练样本,则将随机有放回采样的 m 个样本组成的数据子集用于基准算法的训练。对于每个这样的数据子集,训练数据中的有些样本未被选中、有些样本则多次被选中。平均每个数据子集约占原始训练数据的63%。

在**提升**(boosting)方法中,存在一系列的分类器,每个分类器使用一个加权的样本集。那些被先前分类器误分的样本会获得更大的权值。加权的样本可用于训练基准分类器,也可以影响训练样本的选择(这些样本用于训练未来分类器)。

构建基准分类器的另一种方法是对输入特征进行操纵分解。可以在不同的特征上训练基准分类器。这些特征集通常由手工调整获得。

此外,也可以通过随机化算法的途径获得多种基准分类器。例如,对参数设置不同的初始值可能导致神经网络收敛于不同的局部最小值,进而给出不同的预测值。这些不同的神经网络(参数的初始值不同)可以组合起来进行预测。

7.5 避免过拟合

过拟合的情形会在下述情况中出现:训练数据具有的一些规则并不符合测试数据,而学习者应用这些规则作预测。

【例 7-17】 图 7-13 展示了平方和误差是如何随着线性回归算法迭代次数的变化而变化的。随着迭代次数的增加,训练集上的平方和误差是不断减小的。而对于测试集来说,随着迭代次数的增加,误差先是到达一个最小点,然后又增加了。在决策树学习中,当分裂的次数增加时,会发生相同的行为(变化情况)。

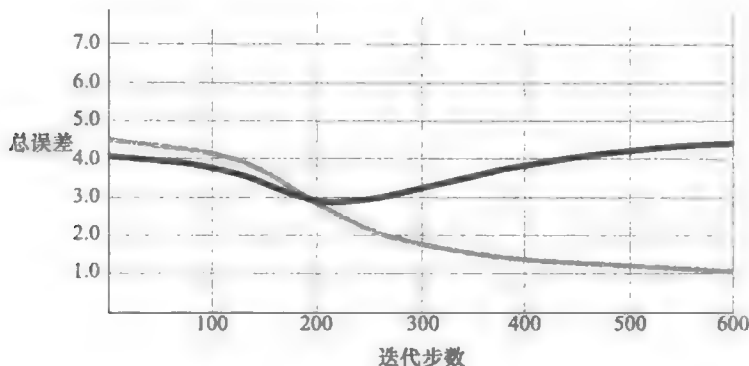


图 7-13 随训练时间而变化的误差。 x 轴是反向传播算法(神经网络带有 3 个隐藏单元)运行的迭代步数。算法采用的训练数据见图 7-9, 测试数据是目标特征值未知的样本集。 y 轴是平方和误差(灰线代表训练集上的误差, 黑线代表测试集上的误差)

320

本节讨论两种避免过拟合的方法。第一种是显式地在模型复杂性和匹配训练数据之间进行折中。第二种是使用部分训练数据检测过拟合。

7.5.1 最大后验概率和最小描述长度

在模型复杂性和匹配训练数据之间进行折中的一种方法是, 在给定数据的条件下选择最有可能的模型。换句话说, 选择最大化条件概率 $P(model|data)$ 的模型, 称该模型为最大后验概率(maximum a posteriori probability)模型, 或 MAP 模型。

可以用贝叶斯规则计算给定数据条件下的模型(或假设)概率:

$$P(model|data) = \frac{P(data|model) \times P(model)}{P(data)} \quad (7.3)$$

似然性 $P(data|model)$ 指的是模型产生这个数据集的概率。如果模型与数据匹配良好, 则似然性的值较大, 反之亦然。先验概率 $P(model)$ 隐含了学习偏置(learning bias), 规定了哪个模型在先验意义上是最可能的。模型的先验概率 $P(model)$ 对于学习偏向选择简单模型是必需的。通常, 简单模型具有更大的先验概率。分母 $P(data)$ 是归一化常数, 以确保概率和为 1。

由于式(7.3)的分母与模型无关, 所以当选择最可能的模型的时候可以忽略它。因此, MAP 模型最大化

$$P(data|model) \times P(model) \quad (7.4)$$

另一种方法是选择最大似然模型(maximum likelihood model)——最大化 $P(data|model)$ 的模型。选择最有可能的模型的问题是, 如果模型空间足够丰富, 则一定存在产生特定数据集的模型, 即 $P(data|model)=1$ 。虽然先验地选择这种模型的可能性非常小, 但我们不想排除这种模型, 因为它有可能是真正的模型。选择最大似然模型等价于选择在假设空间上具有一致先验分布的最大后验概率模型。

1. 决策树的 MAP 学习

为了理解 MAP 学习, 考虑它是如何应用于学习决策树的。如果不存在输入特征值相同而目标特征值不同的样本, 则一定存在完美匹配数据的决策树。如果训练样本没有涵盖输入变量的所有取值, 多棵决策树将完美地匹配数据。然而, 在存在噪声的情况下, 这些决策树都不可能是最好的模型。我们不仅要比较完美匹配数据的模型, 也要比较那些不必

321

如此的模型。MAP 学习提供了比较这些模型的一种方法。

假定存在准确匹配数据的多棵决策树。如果 *model* 代表其中的一棵决策树, 则 $P(\text{data} | \text{model}) = 1$ 。偏好选择这棵而非那棵决策树取决于决策树的先验概率; 先验概率编码了学习偏置。由于简单决策树具有更大的先验概率, 所以偏好简单而非复杂的决策树。

贝叶斯规则提供了一种在简单性和处理噪声数据的能力之间进行折中的方法。通过在叶子节点引入概率, 决策树能够处理噪声数据。当存在噪声的时候, 大的决策树更适合训练数据, 因为它能够考虑训练数据中的随机规律(噪声)。在决策树学习中, 最大似然模型偏爱更大的决策树; 决策树越复杂, 它就越与数据相匹配。先验分布则偏爱更小的决策树。当存在一个决策树上的先验分布的时候, 贝叶斯规则规定了如何在模型复杂性与准确性之间进行折中: 给定数据条件下的模型后验概率和似然性与先验概率的乘积成正比。

【例 7-18】 考虑如图 7-1 所示的数据, 学习器预测用户的行为。

第一棵可能的决策树如图 7-4 左边所示, 称这棵决策树为 d_2 。数据的似然性为 1, 即 $P(\text{data} | d_2) = 1$, d_2 准确地匹配了数据。

第二棵可能的决策树具有如下特点: 没有内部节点, 叶子节点预测 *reads* 的概率是 $\frac{1}{2}$ 。这是在给定数据的前提下, 没有内部节点时的最有可能的决策树, 称之为 d_0 。此时给定模型条件下数据的似然性为

$$P(\text{data} | d_0) = \left(\frac{1}{2}\right)^9 \times \left(\frac{1}{2}\right)^9 \approx 0.000\,001\,49$$

第三棵可能的决策树如图 7-4 右边所示, 它仅仅在特征 *Length* 上作分裂, 叶子节点上的概率分别是 $P(\text{reads} | \text{Length} = \text{long}) = 0$ 和 $P(\text{reads} | \text{Length} = \text{short}) = \frac{9}{11}$ 。注意, $\frac{9}{11}$ 是在训练数据中 *Length* = *short* 时 *reads* 的经验频率。称这棵决策树为 d_{1a} 。此时给定模型条件下数据的似然性为

$$P(\text{data} | d_{1a}) = 1^7 \times \left(\frac{9}{11}\right)^9 \times \left(\frac{2}{11}\right)^2 \approx 0.054\,3$$

第四棵可能的决策树仅仅在特征 *Thread* 上作分裂, 叶子节点上的概率分别是 $P(\text{reads} | \text{Thread} = \text{new}) = \frac{7}{10}$ (10 个样本中, *Thread* = *new* 且 *UserAction* = *reads* 的样本有 7 个) 和 $P(\text{reads} | \text{Thread} = \text{followUp}) = \frac{2}{8}$ 。称这棵决策树为 d_{1b} 。此时给定模型条件下数据的似然性为

$$P(\text{data} | d_{1b}) = \left(\frac{7}{10}\right)^7 \times \left(\frac{3}{10}\right)^3 \times \left(\frac{6}{8}\right)^6 \times \left(\frac{2}{8}\right)^2 \approx 0.000\,025$$

这里列出的仅仅是其中 4 棵可能的决策树。哪一棵最好取决于决策树的先验概率。数据的似然性与决策树先验概率的乘积决定了决策树的后验概率。

2. 描述长度

式(7.4)的负对数(以 2 为底)是

$$(-\log_2 P(\text{data} | \text{model})) + (-\log_2 P(\text{model}))$$

这可以用信息论(information theory)来解释。表达式的左边表示的是用于描述数据(给定模型的条件下)的二进制位数, 右边则表示用于描述模型的二进制位数。最小化上述两项之和的模型称为最小描述长度(minimum description length, MDL)模型。MDL 准则选择

这样的模型，它最小化用于描述模型及其给定模型下的数据的二进制位数。

理解 MDL 准则的一种途径是考虑用尽可能简洁的方法传送数据。模型的使用是为了使传送的路径更短。为了传送数据，首先必须传送模型，然后才是传送数据（在给定模型条件下）。利用一个模型传送数据所需的二进制位，是传送模型和传送数据（在给定模型条件下）所需的二进制位之和。MDL 准则选择利用尽量少的二进制位传送数据的模型。

由于对数函数是单调上升的，所以 MAP 模型和 MDL 模型是等价的。选择具有最大后验概率的模型和选择具有最小描述长度的模型是等价的。

【例 7-19】 例 7-18 没有说明决策树的先验概率。描述长度的概念为定义决策树的先验概率提供了一个基本原则：考虑需要用多少二进制位描述一棵决策树（见例 7-11）。我们必须仔细定义编码，因为每一种编码都应描述一棵决策树；反之，每一棵决策树都应由一种编码描述。

7.5.2 交叉验证

7.5.1 节所述方法的一个问题是，它们要求在 Agent 看到数据之前就拥有关于简洁性的知识。这个问题可理解为：如果 Agent 能够从数据中作出决定，那么一个模型需要多大的复杂性。在学习 Agent 没有关于世界的先验信息的时候，这种方法是适用的。

323

交叉验证 (cross validation) 的思想是将训练集划分为两部分：一部分用于训练，另一部分用作验证集 (validation set)。Agent 利用新的训练集训练模型，在验证集上的预测用于确定采用哪个模型。

考虑图 7-13，训练误差随着决策树规模的增大而减小。交叉验证的思想是选择在验证集上具有最小误差的表示形式。此时，学习过程可以一直进行下去，直到验证集上的误差开始增大。

作为训练数据一部分的验证集和测试集是不一样的。测试集用于评估学习算法的整体性能。将测试集作为学习（训练）的一部分是不可信的。一定要记住，我们的目的是对目前 Agent 还未观察的样本进行预测。测试集是这些还未观察到的样本的一种替代，因此它不能用于训练或验证。

通常，我们想在尽可能多的样本上做训练，因为这样可以获得更好的模型。可是，这样会使得验证集的规模较小，这意味着该验证集与模型的匹配是不确定的：或者良好，或者不好，仅凭运气。存在多种为训练和验证重复使用样本的方法。

k 折交叉验证就是一种确定最优模型复杂性（例如决策树的深度、神经网络中的隐藏节点数等）的方法。 k 折交叉验证 (k -fold cross validation) 将训练集划分成 k 个集合。对于每一种模型的复杂性，学习器训练 k 次，每次使用其中的一个集合作为验证集，其余的用作训练。然后它选择验证集上的平均误差 (k 次运行的平均值) 最小的模型复杂性。它能够返回具有这种复杂性（在所有的数据上训练）的模型。

7.6 基于案例的推理

上述方法设法找到用于未来预测的数据的一个紧致表示。在基于案例的推理 (case-based reasoning) 中，我们存储和访问训练样本（即案例）以解决新问题。为了获得新样本的预测值，用那些与新样本类似或相近的案例预测新样本的目标特征值。这是学习问题的一种极端情形，与决策树和神经网络不同，该方法的离线工作量相对较小，几乎所有的工

作都是在查询时完成的。

324

基于案例的推理可用于分类与回归。它也适用于复杂的案例，例如，在法律案件中，案例指复杂的法律裁决；在规划中，案例指复杂问题的历史解决方案。

如果案例比较简单，采用一种称为 *k* 近邻 (*k*-nearest neighbor) 的算法效果良好。给定一个新样本，与它有相近输入特征值的 *k* 个训练样本用于预测新样本的目标特征值。预测值可以是 *k* 个训练样本目标值的模数、平均值或某种插值，距离近的样本可能比距离远的样本权重更大。

为了采用这种方法，需要一种估量样本之间相似性的距离度量。首先为每个特征的值域定义一个度量，使得特征值转化为可以比较大小的数值。假定 $val(e, X_i)$ 是特征 X_i 在样本 e 上的值的数值表示，则 $(val(e_1, X_i) - val(e_2, X_i))$ 是样本 e_1 和 e_2 在特征 X_i 所定义的维度上的差异。欧氏距离 (Euclidean distance) 指的是所有维度上的差异的平方和的平方根，它可用作两个样本之间的距离度量。一个重要的问题是不同维度的相对比例大小；增加某个维度的比例意味着提高了该特征的重要性。令 w_i 是一个非负实值参数，它表示特征 X_i 的重要性，则样本 e_1 和 e_2 之间的距离可定义为

$$d(e_1, e_2) = \sqrt{\sum_i w_i \times (val(e_1, X_i) - val(e_2, X_i))^2}$$

特征的权重可以通过输入提供。此外，学习这些权重也是可能的。学习 Agent 设法找到最小化误差的参数设置；误差的计算通过预测训练集中的每一个样本 (训练集中的其他样本用作训练) 来实现，称为留一法交叉验证误差 (leave-one-out cross-validation)。

【例 7-20】 考虑在如图 7-1 所示的数据上使用基于案例的推理。不是将数据转化为诸如在决策树或神经网络学习中的间接表示，基于案例的推理直接利用样本预测新样本中用户的行为。

假定学习 Agent 想分类样本 e_{20} ，该样本的 *Author*、*Thread*、*Length* 和 *WhereRead* 特征的值分别是 *unknown*、*followUp*、*short* 和 *home*。首先，学习器设法找到类似的案例。因为存在一个完全匹配的样本 e_{11} ，所以预测用户将采用和样本 e_{11} 相同的行为，即忽略该文章 (*UserAction* = *skips*)。此外，它也可能包含其他相近的样本。

考虑分类样本 e_{19} ，该样本的 *Author*、*Thread*、*Length* 和 *WhereRead* 特征的值分别是 *unknown*、*new*、*long* 和 *work*。此时并不存在完全匹配的样本，因此考虑相近匹配。 e_{19} 和 e_2 、 e_8 、 e_{18} 在特征 *Author*、*Thread* 和 *WhereRead* 上的值相同；和 e_{10} 、 e_{12} 在特征 *Thread*、*Length* 和 *WhereRead* 上的值相同；和 e_3 在特征 *Author*、*Length* 和 *WhereRead* 上的值相同。 e_2 、 e_8 、 e_{18} 预测用户的行为是 *Reads*，其他样本则预测为 *Skips*。预测值究竟应该是什么？决策树认为特征 *Length* 对于预测是最重要的，因此 e_2 、 e_8 、 e_{18} 应该被忽略。对于 sigmoid 线性学习算法，采用例 7-10 中的参数值，预测的结果也是 *Skips*。基于案例的推理算法预测用户的行为之前必须确定各个维度的相对重要性。

325

基于案例的推理需要访问相关案例。*kd* 树 (*kd-tree*) 是对训练样本建立索引的一种方法，目的是快速找到与给定样本相近的训练样本。和决策树类似，*kd* 树在输入特征上作分裂，但其叶子节点是训练样本的子集。由样本集建立一棵 *kd* 树，学习器首先设法找到一个输入特征，将样本集划分为规模大致相同的多个子集；然后分别在多个子集上建立 *kd* 树。这个划分过程一直进行，直到叶子节点上的样本不再变化为止。和决策树一样，新样本可以顺着树向下过滤；正确的匹配将在叶子节点上找到。然而，*kd* 树叶子节点上的样本可能和欲分类的样本相去甚远；它们虽然和顺着树枝向下过滤的值相匹配，但有可

能与其他特征的值不一致。这种树也可以用于搜索那些与测试样本仅有一个特征不相同的样本。见习题 7.16。

基于案例的推理也适用于复杂的案例，例如法律案件和规划问题的历史解决方案。在这种情况下，需要仔细挑选和编辑案例，以使得这些案例发挥作用。基于案例的推理可以看成是下面 4 个任务的一个循环：

检索：给定一个新案例，从案例库中检索类似的案例。

重用：改编检索到的案例以适合这个新案例。

修正：评估答案并根据其效果修改答案。

保存：决定是否将这个新案例保留在案例库中。

修正可涉及其他的一些推理技术，例如，使用推荐的答案作为搜索另一个答案的初始值，或者在交互式系统中人们对案例进行改编。最后保存这个新案例和找到的答案。

【例 7-21】 用户请求解决问题的帮助平台是基于案例的推理系统的一个常见例子。例如，诊断助手可利用基于案例的推理帮助用户在自己的计算机系统上诊断问题。当用户给出问题的描述后，就从案例库中检索出最相近的案例。诊断助手能够推荐一些案例给用户，并根据用户的特殊要求改编每个案例。改变推荐是改编的一个实例，这基于下述问题的考虑：用户拥有什么软件，他们采用什么方法连接因特网，采用的打印机是什么牌子的，等等。如果其中的一个案例可行，那么就将其记录在案例库中，以使得当别的用户询问类似的问题时能突出该案例的重要性。如果没有一个案例是可行的，那么需要采用其他的方法求解问题，例如改编其他案例或由人帮助诊断问题。当问题最终确定后，将相应的案例加入案例库中。

326

7.7 改进假设空间的学习

到目前为止，学习要么是选择最优的表示形式（例如，最优的决策树或神经网络中的最优参数值），要么是依据历史案例预测新案例的目标特征值。本节考虑一种不同的学习概念，即通过勾画与样本一致的假设空间进行学习。目标不是选择一个假设，而是寻找所有一致的假设。这里将阐明学习偏置的作用，并给出对学习算法进行理论分析的一个机制。

我们先给出三个假定条件：

- 只有一个布尔型的目标特征 Y 。这并非完全限定于分类，因为可以利用指示变量将任意的离散型特征转化为布尔型特征。
- 假设给出确定性的预测，即对每个样本给出“真”或“假”的预测，而非概率预测。
- 数据中不存在噪声。

这些假定条件使得用命题的形式定义一个假设成为可能，其中原始命题指的是输入特征的赋值。

【例 7-22】 如图 7-4 所示的决策树可以看成用如下命题定义的 *reads* 表示：

$$pval(e, Reads) = val(e, Short) \wedge (val(e, New) \vee val(e, Known))$$

在本节的其余部分，我们采用一种更简洁的书写形式：

$$reads \leftrightarrow short \wedge (new \vee known)$$

我们的目标是设法找到一个基于输入特征的命题，用以正确地分类样本。

【例 7-23】 考虑交易 Agent 试图基于给定的关键词推断用户将阅读哪些书或文章的问题。假定学习 Agent 拥有下列数据：

文章	Crime	Academic	Local	Music	Reads
a_1	true	false	false	true	true
a_2	true	false	false	false	true
a_3	false	true	false	false	false
a_4	false	false	true	false	false
a_5	true	true	false	false	true

目标是学习用户将阅读哪些文章。

在这个例子中, 目标特征是 *reads*, 学习的目标是找到如下形式的命题:

$$reads \leftrightarrow crime \wedge (\neg academic \vee \neg music)$$

这个命题可以用来分类训练样本以及未来的样本。

假设空间学习假定具有下列集合:

- **实例空间**(instance space) I , 指的是所有可能样本的集合。
- **假设空间**(hypothesis space) \mathcal{H} , 指的是输入特征上的布尔函数集。
- $E \subseteq I$ 是**训练样本**(training example)集。训练样本中的输入特征值和目标特征值均为已知。

如果 $h \in \mathcal{H}$ 和 $i \in I$, 则 $h(i)$ 代表 h 预测样本 i 上的目标特征 Y 的值。

【例 7-24】 在例 7-23 中, I 是 $32(2^5=32)$ 个样本的集合, 每一个样本对应特征取值的一种组合。

假设空间 \mathcal{H} 可以是输入特征的所有布尔组合, 或是施加了某些限制的组合(例如用少于三个特征定义的合取式或命题)。

训练样本 E 是 $\{a_1, a_2, a_3, a_4, a_5\}$, 目标特征是 *Reads*。由于在训练样本中指定了目标特征的值, 且学习器将对未观察到的样本进行预测, 所以学习器需要一个学习偏置。在假设空间学习中, 偏置由假设空间规定。

如果 $\forall e \in E$, h 准确地预测了 e 的目标特征值, 则称假设 h 和训练样本集 E 是一致的(consistent)。换句话说, $h(e) = val(e, Y)$, 即预测值和真正值对于每一个样本而言都是相同的。我们的目标是找到与训练样本一致的 \mathcal{H} 的子集或其中的一个元素。

【例 7-25】 考虑如例 7-23 所示的数据, 并假定 \mathcal{H} 是文字合取的集合。与 $\{a_1\}$ 一致的一个 \mathcal{H} 中的假设是 $\neg academic \wedge music$ 。这意味着用户阅读文章当且仅当 $\neg academic \wedge music$ 为真。这个假设并不是我们想要的, 因为它和 $\{a_1, a_2\}$ 不一致。

7.7.1 变型空间学习

与枚举所有的假设不同, 可以通过在假设空间上规定某些结构而高效地找到与样本一致的 \mathcal{H} 的子集。

如果假设 h_2 蕴含了 h_1 , 则与 h_2 相比, h_1 是一个更一般的假设; 换句话说, 与 h_1 相比, h_2 是一个更特殊的假设。与自身相比, 任意假设均为更一般的假设和更特殊的假设(即两者同时成立)。

【例 7-26】 假设 $\neg academic \wedge music$ 比 *music* 和 $\neg academic$ 都更特殊, 因此, *music* 比 $\neg academic \wedge music$ 更一般。最一般的假设是 *true*, 最特殊的假设是 *false*。

“比……更一般”的关系在假设空间上形成了一个偏序关系。变型空间算法利用这种偏序关系去搜索与训练样本一致的假设。

给定假设空间 \mathcal{H} 和训练样本集 E , **变型空间**(version space)指的是与训练样本一致的

\mathcal{H} 的子集。

变型空间的一般边界(general boundary) G 指的是变型空间中最一般的假设的集合(即没有其他的假设比这些假设更一般)。变型空间的特殊边界(specific boundary) S 指的是变型空间中最特殊的假设的集合。

一般边界和特殊边界的概念是有用的,因为它们完全确定了变型空间:

命题 7.2 给定假设空间 \mathcal{H} 和训练样本集 E ,变型空间可以由它的一般边界和特殊边界导出。特别的,变型空间是假设 h 的集合($h \in \mathcal{H}$),这些假设满足: h 比 S 中的某个元素更一般,比 G 中的某个元素更特殊。

1. 候选删除算法

给定假设空间 \mathcal{H} 和训练样本集 E ,候选删除算法(candidate elimination algorithm)增量地构建变型空间。在该算法中,样本是一个一个加入的:每加入一个样本,都可能因删除与样本不一致的假设而缩减变型空间。对于每一个新样本,候选删除算法通过更新一般边界和特殊边界来达到目的。算法详情见图 7-14。

```

1: procedure CandidateEliminationLearner( $X, Y, E, \mathcal{H}$ )
2:   Inputs
3:      $X$ : 输入特征的集合,  $X = \{X_1, \dots, X_n\}$ 
4:      $Y$ : 目标特征
5:      $E$ : 用于学习的样本集
6:      $\mathcal{H}$ : 假设空间
7:   Output
8:     一般边界  $G \subseteq \mathcal{H}$ 
9:     与  $E$  一致的特殊边界  $S \subseteq \mathcal{H}$ 
10:  Local
11:     $G$ :  $\mathcal{H}$  中的假设集
12:     $S$ :  $\mathcal{H}$  中的假设集
13:    Let  $G = \{true\}$ ,  $S = \{false\}$ ;
14:    for 每一个  $e \in E$  do
15:      if  $e$  是一个正类样本 then
16:        删除  $G$  中将  $e$  分为负类的元素;
17:        删除  $S$  中将  $e$  分为负类的元素  $s$ , 并用一个将  $e$  分为正类的新元素( $s$  的最小推广, 但一般性不如  $G$  中的某个成员)替换它;
18:        删除  $S$  中的非最特殊假设;
19:      else
20:        删除  $S$  中将  $e$  分为正类的元素;
21:        删除  $G$  中将  $e$  分为正类的元素  $g$ , 并用一个将  $e$  分为负类的新元素( $g$  的最小特化, 但比  $S$  中的某个成员更一般)替换它。
22:        删除  $G$  中的非最一般假设

```

图 7-14 候选删除算法

【例 7-27】 考虑候选删除算法是如何处理例 7-23 的数据的, 其中假设空间 \mathcal{H} 是文字合取的集合。

在观察到样本之前, $G_0 = \{true\}$ (用户一定阅读) 和 $S_0 = \{false\}$ (用户一定不阅读)。注意, $true$ 代表空合取, $false$ 代表原子公式与其否定的合取。

当观察到第一个样本 a_1 之后, $G_1 = \{true\}$ 且

$$S_1 = \{crime \wedge \neg academic \wedge \neg local \wedge music\}$$

此时, 最一般的假设是用户一定阅读, 最特殊的假设是只有当输入特征值和样本 a_1 完全一致时用户才阅读。

当观察到第二个样本之后, $G_2 = \{true\}$ 且

$$S_2 = \{crime \wedge \neg academic \wedge \neg local\}$$

此时, 由于 a_1 和 a_2 在特征 *music* 上不一致, 所以得出特征 *music* 无用的结论。

当观察到第三个样本之后, 一般边界变成

$$G_3 = \{crime, \neg academic\}$$

且 $S_3 = S_2$ 。此时存在两个最一般的假设, 即特征 *crime* 为 *true* 时用户一定阅读和特征 *academic* 为 *false* 时用户一定阅读。

当观察到第四个样本之后,

$$G_4 = \{crime, \neg academic \wedge \neg local\}$$

且 $S_4 = S_3$ 。

当所有的 5 个样本均考察完毕之后, 我们有

$$G_5 = \{crime\}$$

$$S_5 = \{crime \wedge \neg local\}$$

此时, 在变型空间中仅存在两个假设。它们之间仅有的区别在于: 当一个样本的 $crime \wedge local$ 为 *true* 时, 它们的预测不一样。如果目标概念可以表示成合取的形式, 仅当一个样本的 $crime \wedge local$ 为 *true* 时才会改变 G 或 S 。这个变型空间可以对所有其他的样本进行预测。

2. 变型空间学习的偏置

回想一下, 对于任意学习算法而言, 偏置对于获得泛化能力是必需的。例 7-27 中必定存在偏置, 因为在仅观察到 16 个输入变量的可能赋值中的 5 个之后, Agent 就能够对还未观察到的样本进行预测。

变型空间学习中的偏置称为语言偏置(language bias)或限制偏置(restriction bias), 因为偏置是通过限制允许的假设而获得的。例如, 一个具有特征 *crime* 为 *false*、*music* 为 *true* 的新样本将被分类为 *false* (用户将不阅读文章), 即使以前并没观察到这样的样本。假设必须是文字的合取这种限制对于预测是足够的。

语言偏置与决策树学习涉及的偏置是不同的。决策树可以表示任意布尔函数。决策树学习涉及偏好偏置(preference bias), 因为它更偏好一些布尔函数: 与大的决策树相比, 它更偏好小的决策树。自顶向下建立单一决策树的决策树学习也涉及搜索偏置(search bias), 因为返回的决策树依赖于所使用的搜索策略。

由于除了涉及选择假设空间 \mathcal{H} 时的语言偏置外, 候选删除算法没有涉及其他的任何偏置, 因此有时它被认为是一种无偏置的学习算法。将变型空间恶化成空集是不难的, 例如使得存在特征 *crime* 为 *false*、*music* 为 *true*, 且用户阅读文章的样本。这意味着目标概念不在 \mathcal{H} 之中。变型空间学习是非抗噪的, 仅仅一个误分的样本就能破坏整个系统。

无偏置(bias-free)的假设空间指的是 \mathcal{H} 由所有的布尔函数组成。在这种情形下, G 总是包含这样的概念: 观察到了所有的负类样本, 其他的均是正类样本。类似的, S 包含所有未观察到的样本均属负类的概念。这种变型空间对于未观察到的样本不能作出任何结论, 因此它没有泛化能力。如果没有语言偏置或偏好偏置, 那么就没有泛化能力, 也就无所谓学习了。

7.7.2 可能近似正确学习

我们已经看到了一些不同的学习算法。本节将讨论一些学习理论方面的问题(属于计算学习理论(computational learning theory)领域)。

计算学习理论致力于回答的一些相关问题如下：

- 增加样本能保证学习器收敛到正确的假设吗？
- 识别一个概念需要多少样本？
- 识别一个概念需要多少计算量？

一般的，第一个问题的答案是否定的，除非能保证样本总会最终排除不正确的假设。可以通过选择一些对于判别假设正确与否没有帮助的样本来误导学习器；因此，如果不能排除这种情况，学习器不能保证一定找到一致的假设。然而，给定一些随机选择的样本，总能选择一致假设的学习器能获得与正确概念任意逼近的解。这需要定义逼近的概念和说明什么是随机选择的样本。

考虑能选择与所有训练样本一致假设的学习算法，并假定存在一个在所有可能样本上的概率分布，训练样本和测试样本均服从该分布。这个概率分布不一定是已知的。我们将证明一个对所有分布都成立的结果。

假设 $h \in \mathcal{H}$ 的误差 (记为 $error(h)$) 定义为选择 I 的一个元素 i , $h(i) \neq val(i, Y)$ 的概率：

$$error(h) = P(h(i) \neq val(i, Y) | i \in I)$$

其中 $h(i)$ 和 $val(i, Y)$ 分别是目标变量 Y 在样本 i 上的预测值和真实值。 I 代表实例空间，即所有可能样本的集合。由于 Agent 通常并不知道 P 或 $val(i, Y)$ ，因此也不知道特定假设的误差。

给定 $\epsilon > 0$ ，如果 $error(h) \leq \epsilon$ ，则称假设 h 是近似正确 (approximately correct) 的。

我们做以下假设。

假设 7.3 训练样本和测试样本是独立地从相同的概率分布中选择的。

样本不能识别与概念相去甚远的假设依然是可能的 (虽然可能性很小)。能选择一个与训练样本一致的假设的学习器，如果对于一个任意的实数 δ ($0 < \delta \leq 1$)，算法不是近似正确的概率至多是 δ ，则该学习器是可能近似正确 (probably approximately correct) 的。换句话说，获得的假设至少是 $1 - \delta$ 概率近似正确的。

332

在假设 7.3 下，对于任意的 ϵ 和 δ ，我们可以保证算法 (返回一致假设的算法) 能至少在 $1 - \delta$ 概率下找到误差小于 ϵ 的假设。而且这个结果不依赖于数据的概率分布。

给定 $\epsilon > 0$ 和 $\delta > 0$ ，将假设空间 \mathcal{H} 分为两部分：

$$\mathcal{H}_0 = \{h \in \mathcal{H} : error(h) \leq \epsilon\}$$

$$\mathcal{H}_1 = \{h \in \mathcal{H} : error(h) > \epsilon\}$$

目标是保证学习器没有选择 \mathcal{H}_1 中的假设的概率大于 δ 。

假设 $h \in \mathcal{H}_1$ ，那么

$$P(\text{对于一个样本而言, } h \text{ 是错误的}) \geq \epsilon$$

$$P(\text{对于一个样本而言, } h \text{ 是正确的}) \leq 1 - \epsilon$$

$$P(\text{对于 } m \text{ 个样本而言, } h \text{ 是正确的}) \leq (1 - \epsilon)^m$$

因此，

$$P(\mathcal{H}_1 \text{ 包含一个对于 } m \text{ 个样本而言是正确的假设})$$

$$\leq |\mathcal{H}_1| (1 - \epsilon)^m$$

$$\leq |\mathcal{H}| (1 - \epsilon)^m$$

$$\leq |\mathcal{H}| e^{-m\epsilon}$$

其中利用了不等式 $(1 - \epsilon) \leq e^{-\epsilon}$, $0 \leq \epsilon \leq 1$ 。

于是，如果 $|\mathcal{H}| e^{-m\epsilon} \leq \delta$ ，我们就能保证 \mathcal{H}_1 没有包含一个对于 m 个样本而言是正确的

假设的概率大于 δ 。因此, \mathcal{H}_0 包含了所有正确的假设的概率大于 $1-\delta$ 。

对 m 求解, 有

$$m \geq \frac{1}{\epsilon} \left(\ln |\mathcal{H}| + \ln \frac{1}{\delta} \right)$$

于是, 我们得到了如下的命题。

命题 7.4 如果一个假设至少与

$$\frac{1}{\epsilon} \left(\ln |\mathcal{H}| + \ln \frac{1}{\delta} \right)$$

个训练样本一致, 那么它至少以 $1-\delta$ 的概率误差小于 ϵ 。

保证上述误差界的样本的数目称为**样本复杂度**(example complexity), 它是 ϵ 、 δ 和假设空间大小的函数。

【例 7-28】 给定假设空间 \mathcal{H} 是 n 个布尔变量上的文字的合取。此时, $|\mathcal{H}| = 3^n + 1$, 这是因为对每一个合取来说, 其中每个变量的状态是下列三种情况之一: 1) 变量本身; 2) 变量的否定; 3) 变量未出现。加 1 是考虑了取值为 *false* 的情形, 即任意原子与其否定的合取。因此, 样本复杂度是 $\frac{1}{\epsilon} (n \ln 3 + \ln \frac{1}{\delta})$, 它是 n 、 $\frac{1}{\epsilon}$ 和 $\ln \frac{1}{\delta}$ 的多项式组合。

假定有 30 个布尔变量, 我们想至少以 99% 的概率保证至多 5% 的误差, 则 $\epsilon = 1/20$ 、 $\delta = 1/100$ 和 $n = 30$ 。这个界表明: 如果找到与 $20 \times (30 \ln 3 + \ln 100) \approx 752$ 个样本一致的假设, 则我们可以保证上述性能。与可能的样本数目 ($2^{30} = 1\,073\,741\,824$) 和假设数目 ($3^{30} + 1 = 205\,891\,132\,094\,650$) 相比, 样本复杂度 (752 个样本) 少得多。

【例 7-29】 如果假设空间 \mathcal{H} 是所有 n 个变量上的布尔函数的集合, 则 $|\mathcal{H}| = 2^{2^n}$; 因此我们需要 $\frac{1}{\epsilon} \left(2^n \ln 2 + \ln \frac{1}{\delta} \right)$ 个样本。样本复杂度是 n 的指数函数。

假定有 30 个布尔变量, 我们想至少以 99% 的概率保证至多 5% 的误差, 则 $\epsilon = 1/20$ 、 $\delta = 1/100$ 和 $n = 30$ 。这个界表明: 如果找到与 $20 \times (2^{30} \ln 2 + \ln 100) \approx 14\,885\,222\,452$ 个样本一致的假设, 则我们可以保证上述性能。

考虑本节开始所提出的第三个问题, 即学习器能够以多快的速度找到可能近似正确的假设? 首先, 如果样本复杂度是参数大小 (例如上述的 n) 的指数函数, 则计算复杂度是指数级的; 这是因为算法必须至少考虑每个样本一次。为了使一个算法具有多项式的时间复杂度, 我们必须找到一个具有多项式样本复杂度的假设空间, 并且该算法处理每个样本的时间复杂度也是多项式的。

7.8 贝叶斯学习

除了选择最可能的模型或者勾画所有与训练数据一致的模型集合之外, 另一种方法是在给定训练样本的条件下计算每个模型的后验概率。

贝叶斯学习 (Bayesian learning) 的思想是: 基于输入特征和所有训练样本, 计算新样本目标特征的条件后验概率。

假定一个新样本的输入特征 $X=x$ 、目标特征是 Y , 目标是计算 $P(Y | X=x \wedge e)$, 其中 e 是训练样本的集合。这是给定特定输入和训练样本的条件下目标变量的概率分布。模型被认为是样本的生成器。令 M 表示不相交且覆盖了全集的模型的集合, 则可通过样本和链式规则推理得到:

$$\begin{aligned} P(Y | x \wedge e) &= \sum_{m \in M} P(Y \wedge m | x \wedge e) = \sum_{m \in M} P(Y | m \wedge x \wedge e) \times P(m | x \wedge e) \\ &= \sum_{m \in M} P(Y | m \wedge x) \times P(m | e) \end{aligned}$$

其中,前面的两个等式是概率论中的定理,最后的等式基于两个假定:模型包含所有关于样本(它们对于特定的预测是必需的)的信息(即 $P(Y|m \wedge x \wedge e) = P(Y|m \wedge x)$);模型不会因为新样本输入的改变而改变(即 $P(m|x \wedge e) = P(m|e)$)。这个公式表明我们调和了所有模型的预测值,其中每个模型用其后验概率(给定样本的条件下)进行了加权。

可以应用贝叶斯规则计算 $P(m|e)$:

$$P(m|e) = \frac{P(e|m) \times P(m)}{P(e)}$$

因此,每个模型的权重取决于该模型预测数据的性能(似然性)和它的先验概率。分母 $P(e)$ 是一个归一化常量,用以确保模型的先验概率之和为 1。当存在许多模型的时候,计算 $P(e)$ 可能非常困难。

给定样本集 $\{e_1, \dots, e_k\}$, 如果对于所有的 i 和 j , 样本 e_i 和 e_j 由模型 m 独立给出, 即 $P(e_i \wedge e_j | m) = P(e_i | m) \times P(e_j | m)$, 则称样本集是独立同分布(independent and identically distributed)的(分布由模型 m 给出)。我们通常假定样本集是独立同分布的。

令 e 代表样本集 $\{e_1, \dots, e_k\}$, 即 e 是 e_i 的合取, 因为所有已观察到的样本均为事实。样本是独立同分布的, 意味着

$$P(e|m) = \prod_{i=1}^k P(e_i|m)$$

除了在参数取值上不同以外, 模型集还可能包含结构不同的模型。贝叶斯学习的一种技巧是显式地表达模型的参数, 并在这些参数上确定分布。

【例 7-30】 考虑不确定情形下的最简单的学习任务。假定存在一个布尔随机变量 Y , 样本的输出为 a 或 $\neg a$ 。目标是给定一些样本学习 Y 上的概率分布。

存在一个决定模型集的参数 ϕ , 假定 ϕ 代表 $Y = \text{true}$ 的概率。我们把这个参数视为在区间 $[0, 1]$ 上的实值随机变量。因此, $P(a|\phi) = \phi$ 和 $P(\neg a|\phi) = 1 - \phi$ 。

假定 Agent 没有关于布尔变量 Y 的概率的先验信息和训练样本之外的知识。这种“无知”可建模为: 将变量 ϕ 的先验概率分布设为区间 $[0, 1]$ 上的均匀分布, 见图 7-15 中标记为 $n_0=0, n_1=0$ 的概率密度函数。

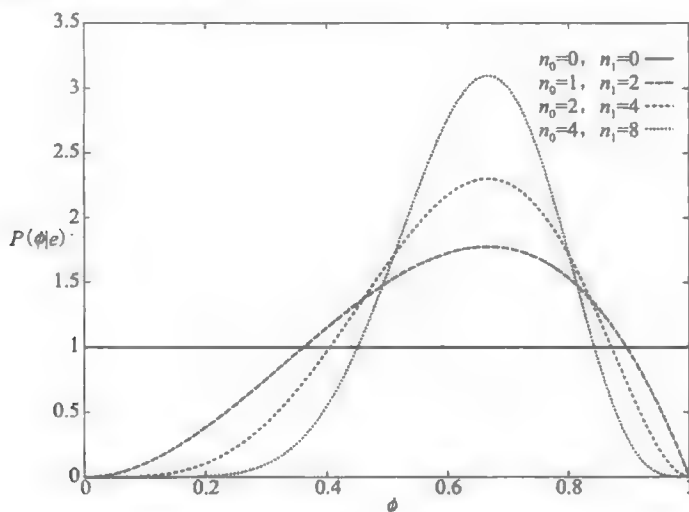


图 7-15 基于不同样本大小的 beta 分布

我们可以通过给定一些样本来更新 ϕ 的概率分布。假定样本(通过多次独立的实验而获得)是输出结果的一个特定序列,其中输出结果由 n_0 个 Y 取 *false* 的样本和 n_1 个 Y 取 *true* 的样本组成。

给定训练样本, ϕ 的后验概率分布可以由贝叶斯规则导出。令 e 是观察(n_1 次 $Y = \text{true}$ 和 n_0 次 $Y = \text{false}$)的特定序列,则由贝叶斯规则可得:

$$P(\phi|e) = \frac{P(e|\phi) \times P(\phi)}{P(e)}$$

其中分母是一个归一化常量,用以确保图 7-15 曲线下方的面积为 1。

假定样本是独立同分布的,则

$$P(e|\phi) = \phi^{n_1} \times (1-\phi)^{n_0}$$

这是由于存在 n_0 个 $Y = \text{false}$ 的样本(其中每个样本出现的概率是 $1-\phi$)和 n_1 个 $Y = \text{true}$ 的样本(其中每个样本出现的概率是 ϕ)。

先验概率 $P(\phi)$ 可能是区间 $[0, 1]$ 上的均匀分布。当 Agent 没有关于概率的先验信息时,均匀分布的假定是合理的。

图 7-15 给出了一些基于不同样本大小的变量 ϕ 的后验分布,其中假定 $P(\phi)$ 为均匀分布。图中有 3 种情形: ($n_0=1, n_1=2$)、($n_0=2, n_1=4$)、($n_0=4, n_1=8$)。每一种情形均在相同的位置(即 $\frac{2}{3}$)达到峰值。更多的样本使得曲线图更为尖凸。

上述例子中的分布即是所谓的 **Beta 分布** (Beta distribution), 它的参数包括两种样本计数($Y = \text{false}$ 和 $Y = \text{true}$) α_0 和 α_1 , 以及一个概率 p 。按照惯例, 参数 α_i 的取值比样本计数多 1, 即 $\alpha_i = n_i + 1$ 。Beta 分布定义为

$$\text{Beta}^{\alpha_0, \alpha_1}(p) = \frac{1}{K} p^{\alpha_1-1} \times (1-p)^{\alpha_0-1}$$

其中 K 是一个归一化常量,用以确保所有值上的积分为 1。因此, $[0, 1]$ 上的均匀分布是 beta 分布 $\text{Beta}^{1,1}$ 。

将 Beta 分布进行推广,使其包含的参数多于两个,就得到了所谓的 **Dirichlet 分布** (Dirichlet distribution)。Dirichlet 分布包括两种类型的参数,即“计数” $\alpha_1, \dots, \alpha_k$ 和概率参数 p_1, \dots, p_k , 形式如下:

$$\text{Dirichlet}^{\alpha_1, \dots, \alpha_k}(p_1, \dots, p_k) = \frac{1}{K} \prod_{j=1}^k p_j^{\alpha_j-1}$$

其中, K 是一个归一化常量,用以确保所有值上的积分为 1; p_i 是第 i 个输出的概率($0 \leq p_i \leq 1$); α_i 比第 i 个输出的计数多 1, 即 $\alpha_i = n_i + 1$ 。如果从每一维的角度看, Dirichlet 分布函数的图像与图 7-15 类似(即每一个 P_i 在 0 到 1 之间变化)。

在大多数情况下,将所有模型(用其后验分布加权)进行累加是困难的,因为模型可能比较复杂(例如它们是决策树,甚至是信念网络)。然而,对于 Dirichlet 分布来说,第 i 个输出的期望值是:

$$\frac{\alpha_i}{\sum_j \alpha_j}$$

参数 α_i 比第 i 个输出的计数多 1 的原因就是为了使得上述式子比较简单。仅当所有的 α_i 非负和并非所有的 α_i 都为 0 时,上述定义才有意义。

【例 7-31】 考虑例 7-30,它基于一个观察序列(由 n_0 个 Y 取 *false* 的样本和 n_1 个 Y 取 *true* 的样本组成)确定 ϕ 的值。考察如图 7-15 所示的后验概率分布,令人感兴趣的是,虽

然 ϕ 的最可能的后验概率值是 $\frac{n_1}{n_0+n_1}$ ，期望值却是 $\frac{n_1+1}{n_0+n_1+2}$ 。

因此， $(n_0=1, n_1=2)$ 、 $(n_0=2, n_1=4)$ 和 $(n_0=4, n_1=8)$ 三种情形的期望值分别是 $\frac{3}{5}$ 、 $\frac{5}{8}$ 和 $\frac{9}{14}$ 。随着学习器获得的样本数不断增加，期望值趋近于 $\frac{n}{m}$ 。

这种估计值比 $\frac{n}{m}$ 更好的原因如下。首先，它告诉我们如果学习 Agent 没有样本该如何处理：使用先验的均匀分布，期望值取 $\frac{1}{2}$ 。这就是 $n=0$ 、 $m=0$ 情形下的期望值。第二，考虑 $n=0$ 、 $m=3$ 的情形。Agent 不应该使用 $P(y)=0$ ，因为它表明 Y 是不可能的，这当然是错误的！使用先验的均匀分布，这种情形下的期望值 $\frac{1}{5}$ 。

Agent 搜索最优的模型并非一定要从先验的均匀分布开始，它可以从任意的先验分布开始。如果 Agent 从 Dirichlet 分布开始搜索，则模型的后验概率将服从 Dirichlet 分布。可以通过将观察到的计数值加到先验分布的参数 α_i 上来得到该分布。

独立同分布假设可以表示成一个信念网络，其中每个 e_i 在给定模型 m 的条件下是独立的。这种独立性假设可以表示成如图 7-16 所示的信念网络。如果将 m 转化成一个离散变量，则可利用第 6 章所介绍的任意一种推理方法在该网络中进行推理。该网络中的一种标准的推理技术是基于所有观察到的 e_i 去查询模型变量或未观察到的 e_i 变量。

为学习问题指定一个信念网络的缺陷是该模型会随着观察样本数的增加而变得庞大。可以在接收到观察样本之前指定信念网络，其中接收观察样本使用一个称为盘子模型 (plate model) 的方法。盘子模型规定在该模型中将使用什么变量和什么样本将会被反复观察。图 7-16 右侧展示了一个盘子模型，它表示了与图中左侧的信念网络相同的信息。盘子用一个矩形表示，其中包含一些节点和索引号 (在盘子底侧右边)。用索引号对盘中的节点进行索引。在盘子模型中，盘中变量有多份副本，一个副本对应索引的一个值。直观的想法是存在一堆盘子，每个盘子对应索引的一个值。盘子的数量依赖于观察的样本数和查询的内容。在图 7-16 中，盘中的所有节点共享一个公共的父节点。给定父节点条件下，盘中变量的每一个副本的概率与对应索引的概率是一样的。

盘子模型使得我们可以说明变量之间更为复杂的关系。在层次贝叶斯模型 (hierarchical Bayesian model) 中，模型的参数可依赖于其他的参数 (正是这种意义上，称之为层次模型)。

【例 7-32】 假定诊断助手 Agent 想对一个特定住院病人患流感 (在观察到症状之前) 的概率进行建模。可以将病人的这种先验信息和观察到的症状结合起来做诊断。Agent 的目标是根据有关病人 (包括相同医院和不同医院内的病人) 的统计资料学习这种概率。这个问题包括从“存在许多关于该医院的数据 (应该会使用这些数据)”到“不存在病人所在医院的有关数据”的各种情形。层次贝叶斯模型可用于组合这些统计数据。

假定对于一个在医院 H 住院的病人 X ，存在一个随机变量 S_{HX} ，它在病人患流感时取真值 (假定医院可根据身份证号唯一地确定每个病人)。每个医院 H 都存在一个值 ϕ_H ；对于医院 H 中的每个病人， ϕ_H 用于表示患流感的先验概率。在贝叶斯模型中， ϕ_H 是区间 $[0, 1]$ 上的一个实值随机变量。 S_{HX} 依赖于 ϕ_H ($P(S_{HX} | \phi_H) = \phi_H$)。假定 ϕ_H 服从 Beta 分



图 7-16 贝叶斯学习的信念网络和盘子模型

337

338

布, ϕ_{h_1} 和 ϕ_{h_2} 之间是相互独立的, 但均依赖于超参数(hyperparameter)。超参数可以是先验的样本计数 α_0 和 α_1 。参数依赖于超参数: $P(\phi_{h_i} | \alpha_0, \alpha_1) = \text{Beta}^{\alpha_0, \alpha_1}(\phi_{h_i})$; α_0 和 α_1 是实值随机变量, 确定它们的值需要某种先验分布。

盘子模型和相应的信念网络见图 7-17。图 7-17a 是盘子模型, 其中为每所医院维护一个外部盘子的副本, 为每个住院病人维护一个内部盘子的副本。图 7-17b 是信念网络。 S_{HX} 的部分观察值将影响 ϕ_H , 进而影响 α_0 和 α_1 ; α_0 和 α_1 反过来将影响其他的 ϕ_H 变量和未观察到的 S_{HX} 变量。

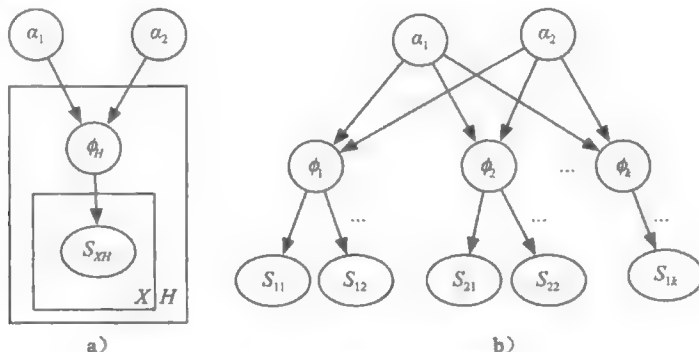


图 7-17 层次贝叶斯模型

现存的统计方法可用于估算该网络。然而, 如果变量是离散型的, 则可应用第 6 章介绍的任意方法。

除了利用 ϕ 的后验概率分布导出期望值, 我们还可以利用它回答其他的问题, 例如: ϕ 的后验概率分布在区间 $[a, b]$ 上的概率是多少? 即计算 $P((\phi \geq a \wedge \phi \leq b) | e)$ 。这就是牧师托马斯贝叶斯在 200 多年以前解决的问题[Bayes, 1763], 他给出的答案(虽然使用了比较繁冗的记号)是:

$$\frac{\int_a^b p^n \times (1-p)^{n-1} dp}{\int_0^1 p^n \times (1-p)^{n-1} dp}$$

这种知识可用于测量中, 可能会有这样的报告: 20 次测量中, 至少有 19 次测量的误差至多不超过 5%。这与用于可能近似正确(PAC)学习中的信息类型一致, PAC 学习保证误差至少以 $1-\delta$ 的概率至多不超过 ϵ 。如果 Agent 选择 $[a, b]$ 的中点(即 $\frac{a+b}{2}$)作为它的假设,

则误差小于或等于 $\frac{b-a}{2}$, 这种结论仅当假设位于 $[a, b]$ 区间内才成立。 $1-\delta$ 的值相当于

$P((\phi \geq a \wedge \phi \leq b) | e)$ 。如果 $\epsilon = \frac{b-a}{2}$ 和 $\delta = 1 - P((\phi \geq a \wedge \phi \leq b) | e)$, 选择中点作为假设将导

致误差至少以 $1-\delta$ 的概率至多不超过 ϵ 。PAC 学习给出了最坏情况下的结果, 而贝叶斯学习则给出了期望值。通常情况下, 贝叶斯估计更准确, 但 PAC 结果给出了误差的界。与 PAC 学习相比, 贝叶斯学习所需的样本复杂度(见 7.7.2 节)通常少很多——与保证获得满意的准确率相比, 只需要更少的样本就可以期望获得满意的准确率。

7.9 本章小结

- 学习是一个 Agent 基于经验改善其行为的能力。
- 有监督学习指的是给定一组由输入-输出对组成的训练样本, 预测一个新输入所对应的输出。

- 给定一些训练样本, Agent 构建一种能用于新预测的表示形式。
- 线性分类器、决策树和贝叶斯分类器均为简单的学习模型, 它们是更复杂模型的基础。
- 给定训练样本的条件下, Agent 能够选择最优的假设, 能够勾画与训练数据一致的所有假设, 或者计算假设的后验概率。

339
340

7.10 参考文献及进一步阅读

介绍机器学习的一些比较好的书籍见 Mitchell[1997], Duda、Hart 和 Stork[2001], Bishop[2008]以及 Hastie、Tibshirani 和 Friedman[2009]。

Shavlik 和 Dietterich[1990]编辑的论文集中包含许多经典的学习论文。Michie、Spiegelhalter 和 Taylor[1994]给出了多种学习算法在许多不同问题上的实验评估。Briscoe 和 Caelli[1996]讨论了多种不同的机器学习算法。Weiss 和 Kulikowski[1991]综述了分类学习的一些技术。Davis 和 Goadrich[2006]讨论了精确率、召回率和 ROC 曲线。

Spiegelhalter、Franklin 和 Bull[1990]提出了组合专家知识和数据的方法。

Quinlan[1986]讨论了决策树学习。Quinlan[1993]介绍了一种成熟的决策树学习工具。Gini 指数(习题 7.10)是用于 CART 算法[Breiman, Friedman, Olshen 和 Stone, 1984]的特征分裂标准。

Friedman、Greiger 和 Goldszmidt[1997]描述了 TAN 网络。Zhang[2004]描述了潜在树模型。

神经网络的综述见 Bishop[1995], Hertz、Krogh 和 Palmer[1991], Jordan 和 Bishop[1996]。Rumelhart、Hinton 和 Williams[1986]介绍了反向传播算法。Minsky 和 Papert[1988]分析了神经网络的局限性。

集成学习的综述见 Dietterich[2002]。Schapire[2002], Meir 和 Rätsch[2003]描述了“提升”方法。

基于案例的推理的综述见 Aamodt 和 Plaza[1994], Kolodner 和 Leake[1996], Lopez De Mantaras、Mcsherry、Bridge、Leake、Smyth、Craw、Faltings、Maher、Cox、Forbus、Keane、Aamodt 和 Watson[2005]。最近邻算法的综述见 Duda 等[2001]和 Dasarathy[1991]。维度(特征)加权最近邻学习算法见 Lowe[1995]。Riesbeck 和 Schank[1989]给出了基于案例的推理的一个经典的介绍, 最近的介绍见 Aha、Marling 和 Watson[2005]。

Mitchell[1977]定义了变型空间。Valiant[1984]介绍了 PAC 学习。本章有关 PAC 学习的分析应归功于 Haussler[1988]。Kearns 和 Vazirani[1994]给出了一个关于计算学习理论和 PAC 学习的很好的介绍。更多的关于变型空间和 PAC 学习的细节见 Mitchell[1997]。

贝叶斯学习的综述见 Jaynes[2003], Loredó[1990], Howson 和 Urbach[2006], Cheeseman[1990]。也可以参考关于贝叶斯统计的书, 例如 Gelman、Carlin、Stern 和 Rubin[2004], Bernardo 和 Smith[1994]。Buntine[1992]描述了决策树的贝叶斯学习。Grünwald[2007]讨论了最小描述长度(MDL)原理。

341

关于机器学习的研究成果, 参考期刊《机器学习研究》(Journal of Machine Learning Research (JMLR))、《机器学习》(Machine Learning)和每年的国际机器学习会议(International Conference on Machine Learning, ICML)、神经信息处理协会的论文集(Proceedings of the Neural Information Processing Society, NIPS)、或一般的人工智能期刊, 例如《人工智能》(Artificial Intelligence)和《人工智能研究》(Journal of Artificial Intelligence Research), 以及其他许多专业的会议或期刊。

7.11 习题

7.1 该习题的目的是填充如图 7-3 所示的表。

- 证明关于训练数据的最优预测。为此, 分别找出绝对误差、平方和误差和熵的最小值, 以及最大化似然性的值。最大或最小值在端点或导数为零的点取得。
- 为了确定测试数据的最优预测, 假定数据是根据某个真实的参数 p_0 随机生成的。对于 $p_0 \in [0, 1]$ 的不同取值, 设法进行下列操作: 通过采样(采样的概率为 p_0)生成 k 个训练样本(尝试不同的 k , 一些比较小, 例如 5; 一些比较大, 例如 1000)。由生成的样本可以计算 n_0 和 n_1 。利用相同的参数 p_0 生成一个包含许多测试样本的测试集。对于每一种优化准则(绝对误差、平方和

误差和似然性(或熵)), 下列哪种情形在测试集上的误差较小?

i) 模数

ii) $n_1/(n_0+n_1)$

iii) 如果 $n_1=0$, 则使用 0.001; 如果 $n_0=0$, 则使用 0.999; 否则使用 $n_1/(n_0+n_1)$ 。(当计数为 0 时用不同的数进行尝试。)

iv) $(n_1+1)/(n_0+n_1+2)$

v) $(n_1+\alpha)/(n_0+n_1+2\alpha)$ (尝试不同的 $\alpha>0$ 取值)

vi) 别的预测器, 它是 n_0 和 n_1 的函数。

你可能不得不为每种参数值生成许多不同的训练集。(基于这些数学意义上随机生成的样本, 对于每一种准则, 你能证明最优的预测器是什么?)

- 7.2 在值域为 $\{0, 1\}$ 的特征(无输入特征)的点估计中, Agent 基于参数 $p \in [0, 1]$ 做随机预测是可能的, 其中预测为 1 和 0 的概率分别是 p 和 $1-p$ 。对于下面的每一种误差度量, 给出训练集上的期望误差, 其中训练集由 n_0 个取值为 0 的样本和 n_1 个取值为 1 的样本构成, n_0 和 n_1 均为 p 的函数。最小化误差的 p 值是多少? 与如图 7-3 所示的预测值相比, p 的值是否更好?

(a) 绝对误差

(b) 平方和误差

(c) 最坏情况误差

- 7.3 假定存在这样一个系统, 它观察用户看电视的习惯, 目的是向该用户推荐他可能喜欢的电视节目。每个节目均由下列特征描述: 是否是喜剧? 有医生吗? 有律师吗? 有枪支吗? 给定如图 7-18 所示的关于用户是否喜欢电视节目的样本, 我们的目标是利用这些数据学习特征 *Likes* 的值(即基于电视节目的特征预测用户喜欢哪个电视节目)。

你可以在网站 AIspace.org 上找到关于该作业的有用的小程序。(在开始之前, 看看你是否能够找出用户喜欢的电视节目的规律。)

示例	Comedy	Doctors	Lawyers	Guns	Likes
e_1	false	true	false	false	false
e_2	true	false	true	false	true
e_3	false	false	true	true	true
e_4	false	false	true	false	false
e_5	false	false	false	true	false
e_6	true	false	false	true	false
e_7	true	false	false	false	true
e_8	false	true	true	true	true
e_9	false	true	true	false	false
e_{10}	true	true	true	false	true
e_{11}	true	true	false	true	false
e_{12}	false	false	false	false	false

图 7-18 习题 7.3 的训练样本

(a) 采用绝对误差, 给出只有一个节点(即无分裂)的最优决策树。该决策树的误差是多少?

(b) 采用平方和误差, 求解与(a)相同的问题。

(c) 采用绝对误差, 给出深度为 2(即根节点是唯一的有孩子的节点)的最优决策树。对于树中的每个叶子节点, 给出经过滤到达该节点的样本。该决策树的误差是多少?

(d) 采用平方和误差, 求解与(c)相同的问题。

(e) 正确分类所有训练样本的最小决策树是什么? 自顶向下每一步优化信息增益的决策树与正确分类所有训练样本的最小决策树表示相同的函数吗?

(f) 给出两个不在图 7-18 中的样例, 并展示如何应用最小决策树对该样例进行分类。然后解释决策树固有的学习偏置。(偏置是如何指导这些特定预测的?)

(g) 这个数据集是线性可分的吗? 解释原因。

- 7.4 考虑如图 7-5 所示的决策树学习算法和如图 7-1 所示的数据。假定停止的准则是所有样本的分类不再发生变化。如图 7-4 所示的决策树是通过选择具有最大信息增益的特征作分裂而构建的。问题是考虑当选择不同的特征作分裂时会发生什么。

(a) 改变算法使得总是选择特征列表中的第一个元素作分裂。当特征按顺序[Author, Thread, Length, WhereRead]排列时生成的决策树是什么? 与选择具有最大信息增益的特征作分裂得到的决策树相比, 新的决策树表示一个不同的函数吗? 解释原因。

(b) 当特征按顺序[WhereRead, Thread, Length, Author]排列时生成的决策树是什么? 与选择具有最大信息增益的特征作分裂得到的决策树相比或与(a)中的决策树相比, 新的决策树表示一

个不同的函数吗？解释原因。

(c) 是否存在这样的一棵决策树：它正确分类了训练样本，但与采用前面的算法得到的决策树相比却表示一个不同的函数？如果有，给出这样的决策树；如果没有，解释原因。

7.5 考虑式(7.1)，它给出了线性预测的误差。

(a) 当 $n=1$ (即只有一个输入特征) 时，给出最小化误差时的权值计算公式。(提示：对每一个权重变量求导，并令其为 0。)

(b) 当 n 取任意值时，给出最小化误差时的权值的计算公式集。

(c) 当使用 sigmoid 线性函数(激活函数为 sigmoid 或 logistic 函数的扁线性函数)时，为什么解析地最小化误差是困难的？

7.6 假定对于神经网络的输出，我们规定大于 0.5 的值为真，小于 0.5 的值为假(在输入激活函数之前，任意正数为真、负数为假)。

对于具有两个隐藏节点的神经网络，在如图 7-9 所示的数据上运行 AISpace.org 上的神经网络学习小程序。获得最终的参数值后，给出一个表示布尔函数的逻辑公式(或一棵决策树或一个规则的集合)，其中的布尔函数计算隐藏节点和输出节点的值。该公式或规则集不应是任意实数。

(提示：一种“蛮力”搜索法是：对于每一个隐藏节点，检查输入值的 16 种组合；然后确定输出的真正值。更好的方法是设法理解函数本身。)

神经网络学得了与决策树相同的函数吗？

7.7 确定决策树的空间大小。假定学习问题中存在 n 个二元特征。总共有多少棵不同的决策树？这些决策树表示了多少个不同的函数？两棵不同的决策树可能会导致相同的函数吗？

7.8 扩展如图 7-5 所示的决策树学习算法，使得可以表示多值特征，并返回决策树的规则形式。

344

需要克服的一个问题是，如何处理没有样本与所选特征的特定值相符的情形。此时，你必须作出一个合理的预估。

7.9 如图 7-5 所示的决策树学习算法在特征用完后必须停止，即使并非所有的样本在分类上取得一致。假定你正在构建一棵决策树，且到达了这样一个阶段：已经没有剩余特征可用于分裂，但训练集中还有 n_1 个正类样本和 n_0 个负类样本没有处理完毕。此时有 3 种建议的策略可用：

i) 返回多数样本的取值——如果 $n_1 > n_0$ ，则返回 *true*；如果 $n_1 < n_0$ ，则返回 *false*；如果 $n_1 = n_0$ ，则返回 *true* 和 *false* 中的任意一个。

ii) 返回经验频率 $n_1 / (n_0 + n_1)$ 。

iii) 返回 $(n_1 + 1) / (n_0 + n_1 + 2)$ 。

对于下列定义的误差，上述哪种策略在训练集上的误差最小？解释原因。

(a) 样本的值($1 = \text{true}$ 和 $0 = \text{false}$)与决策树预测值($1 = \text{true}$ 、 $0 = \text{false}$ 或概率值)之差的和。

(b) 样本的值($1 = \text{true}$ 和 $0 = \text{false}$)与决策树预测值($1 = \text{true}$ 、 $0 = \text{false}$ 或概率值)之差的平方和。

(c) 数据的熵。

7.10 关于在决策树搜索中选择哪个特征进行分裂的问题，除了如 7.3.1 节所述的最大化信息分裂的方法，另一种可选的启发式方法是使用 Gini 指数。

一个样本集的 Gini 指数(关于目标特征 Y)是样本集不纯性的一种度量：

$$\text{gini}_Y(\text{Examples}) = 1 - \sum_{\text{val}} \left(\frac{|\{e \in \text{Examples}; \text{val}(e, Y) = \text{Val}\}|}{|\text{Examples}|} \right)^2$$

其中 $|\{e \in \text{Examples}; \text{val}(e, Y) = \text{Val}\}|$ 是特征 Y 的值为 Val 的样本个数， $|\text{Example}|$ 是所有样本的总个数。Gini 指数总是非负的，仅当所有样本具有相同特征值的时候才为 0。当样本在特征的不同取值上平均分布时，Gini 指数达到最大值。

选择哪个特征作分裂的启发式方法是：选择能最小化训练样本在目标特征上总的纯度(即所有叶子节点不纯度的和)的特征。

(a) 实现使用 Gini 指数的决策树搜索算法。

(b) 尝试在一些数据库上分别运行基于 Gini 指数的算法和基于最大化信息分裂的算法，看看哪个

结果更优。

(c) 找出一个样例数据库, 使得基于 Gini 指数的算法找到一个与基于最大化信息增益的启发式算法不同的决策树。对于这个数据库, 哪种启发式算法更优? 并考虑哪种启发式算法对手头的数据库更为敏感。

(d) 设法找出一个样例数据库, 使得基于最大化信息分裂的算法比基于 Gini 指数的算法更为敏感。设法找出另一个样例数据库, 使得基于 Gini 指数的算法更优。(提示: 尝试极端分布。)

7.11 参考例 7-18 的说明, 为描述决策树定义一种编码。确保每一种编码对应一棵决策树(对于足够长的二进制位序列, 初始的分段将描述一棵唯一的决策树), 每一棵决策树都有一种编码。这种编码如何翻译为决策树上的先验分布? 特别的, 引入一个新的分裂有多大的可能性必须补偿分裂的先验概率的减少值(假定在编码中小的决策树比大的决策树更易于描述)?

7.12 说明梯度下降搜索如何应用于学习一个最小化绝对误差的线性函数。(提示: 对误差做实例分析。除了误差为 0 的情形(此时不必进行更新操作), 该误差在各个点都是可微的。)

7.13 给出一个实例, 使得当使用经验频率作为概率时, 朴素贝叶斯分类器能给出不一致的结果。(提示: 你需要两个特征, 比方说 A 和 B , 和一个值为 $\{0, 1\}$ 二元分类, 比方说 C 。构造一个数据集, 其中经验频率为 $P(a|C=0)=0$ 和 $P(b|C=1)=0$ 。)什么样的观察与该模型不一致?

7.14 在如图 7-1 所示的数据上运行 AIspace.org 上的神经网络学习器。

(a) 假定你决定将神经网络的任意大于 0.5 的预测值视为真, 小于 0.5 的视为假。初始时有多少被误分的样本? 40 次迭代之后又有多少被误分的样本? 80 次迭代之后呢?

(b) 采用相同的样本和相同的初始值, 改变梯度下降搜索的步长。至少尝试 $\eta=0.1$ 、 $\eta=1.0$ 和 $\eta=5.0$ 三种情况。对步长和算法收敛之间的关系作出评论。

(c) 找到最终的参数值之后, 给出每个单元计算的逻辑公式。你可以这样做: 首先考虑每个单元输入值的真值表和确定每一个组合的输出, 然后约简公式。总是可以找到这样的公式吗?

(d) 不同的参数设定的初始值是不同的。如果为所有的参数设定相同的(随机)值, 将会发生什么情况? 在其他的数据上进行相同的测试, 并猜测一般情况下会发生什么。

(e) 评论神经网络算法的下列停止准则:

i) 有限步的迭代, 其中有限步数是初始时设定的。

ii) 平方和误差小于 0.25。解释为什么 0.25 可能是一个合适的选择。

iii) 所有导数的绝对值均小于 $\epsilon(\epsilon>0)$ 。

iv) 将数据划分为训练集和测试集, 当在测试集上的误差增加时算法停止。

你期望哪种准则更好地处理“过拟合”问题? 哪种准则能确保梯度下降法停止搜索? 哪种准则能确保如果停止搜索, 学得的网络能准确地对测试数据进行预测?

7.15 神经网络学习算法是基于每个样本更新参数值的。为了准确地计算导数值, 应该在观察到所有样本之后才进行参数值的更新。实现这样的学习算法, 并与增量算法进行收敛速度和算法运行速度方面的比较。

7.16 (a) 画出如图 7-1 所示数据的一棵 kd 树。用于分裂的最优特征应当最大程度上将样本集划分为大小相同的两类。假定你知道在后续的查询中不会出现特征 *UserAction*, 因此不应选它作分裂。说明训练样本分别位于哪个叶子节点上。

(b) 对于一个特征 *Author*、*Thread*、*Length* 和 *WhereRead* 的值分别是 *unknown*、*new*、*long* 和 *work* 的新样本, 指出与它最相似的训练样本在决策树上的位置。

(c) 基于这个实例, 讨论从 kd 树的查找表会返回哪些样本。这为什么和决策树的查找表不同?

7.17 实现一个将训练样本存储在一棵 kd 树上的最近邻学习系统, 使用那些具有不同最小特征(这些特征的权重是均衡的)数目的近邻样本作决策。在实践中这种方法效果如何?

确定性规划

人们每天早晨都会为一天的行程做一个规划，并执行这个规划，这个规划伴随他度过一天天忙碌的生活。如果事先不做这个规划，就只能根据事件发生的几率随机支配时间，那么生活很快就会陷入混乱。

——Victor Hugo(1802—1885)

规划是 Agent 如何实现其目标的行动方案。Agent 要完成任何事情即使是实现最简单的目标，它必须推测(考虑)其未来的目标。因为 Agent 通常不能在一步之内就实现其目标，所以它在任一时刻的行为取决于它将来的行为。而它将来的行为又取决于现在的状态，也就是它过去的行为。本章介绍了如何表示 Agent 的动作以及执行动作后所产生的结果，以及 Agent 如何用这些模型制定一个规划实现其目标。

本章主要考虑以下几种情况：

- Agent 的行动是确定性的，即 Agent 能够预测它的行动的结果。
- 不存在超出 Agent 控制的改变世界状态的外部事件。
- 世界是完全可观察的，Agent 可以观察世界的当前状态。
- 时间是一个状态到另一个状态的离散推移。
- 目标是必须实现或者维护的状态谓词。

在后面章节中，这些假设条件有所放宽。

349

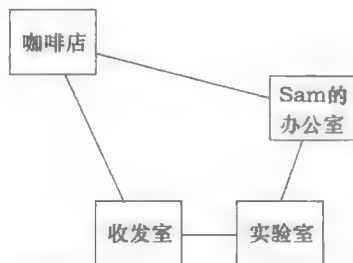
8.1 状态、动作以及目标的表示

为了要推测(考虑)做什么，Agent 必须要有目标、某个世界模型，以及一个行动结果模型。

确定性动作是状态到状态的一个部分函数。之所以是部分函数，是因为在每一个状态下不是每个动作都会执行。例如，机器人如果不在物体的附近，那么它就不能执行拿起这个物体的动作。动作的**先决条件**(precondition)是指何时能够执行这个动作，而动作的**影响**(effect)是指执行动作后产生的状态，即结果状态。

【例 8-1】 考虑传送邮件和咖啡的传送机器人世界。假设一个简单的环境有 4 个位置，如图 8-1 所示。机器人 Rob 可以在咖啡店买咖啡，在收发室取邮件，移动，还可以传送咖啡和邮件。把咖啡送到 Sam 的办公室的动作将会中止 Sam 想要咖啡的动作。收发室有要送往 Sam 办公室的邮件。这个环境非常简单，但是也足够说明表示动作方面以及规划中的许多问题了。

状态可描述为以下特征：



描述状态的特征

RLoc—Rob 的位置

RHC—Rob 是否有咖啡

SWC—Sam 是否想要咖啡

MW—是否有邮件

RHM—Rob 是否有邮件

动作

mc—顺时针移动

mcc—逆时针移动

puc—取咖啡

dc—递咖啡

pum—取邮件

dm—递邮件

图 8-1 传送机器人领域

350

- 机器人的位置($RLoc$)可以是咖啡店(cs)、Sam 的办公室(off)、收发室(mr)或者实验室(lab)中的某个。
- 机器人是否有咖啡(RHC): rhc 表示 Rob 有咖啡, \overline{rhc} 表示 Rob 没有咖啡。
- Sam 是否想要咖啡(SWC): swc 表示 Sam 想要咖啡, \overline{swc} 表示 Sam 不想要咖啡。
- 收发室是否有邮件(MW): mw 表示有邮件, \overline{mw} 表示没有邮件。
- 机器人身上是否有邮件(RHM): $rh m$ 表示机器人身上有邮件, $\overline{rh m}$ 表示机器人身上没有邮件。

假设 Rob 有 6 个动作:

- Rob 可以顺时针移动(mc)。
- Rob 可以逆时针移动(mcc)。
- Rob 在咖啡店可以拿起咖啡。 puc 表示 Rob 拿起咖啡, 它的前提条件是 $\overline{rhc} \wedge RLoc = cs$, 也就是说当 Rob 的位置是 cs 并且身上还没有咖啡的状态下可以拿起咖啡。这个动作的结果就是使得 RHC 为真。它不会影响其他的特性。
- 如果 Rob 拿着咖啡并且在 Sam 的办公室, Rob 就可以递送咖啡。 dc 表示 Rob 递咖啡, 它的前提条件是 $rhc \wedge RLoc = off$ 。这个动作的结果就是使得 RHC 为真并且 SWC 为假。
- 如果 Rob 在收发室并且收发室有邮件, Rob 就可以取邮件了。 pum 表示 Rob 取邮件。
- 如果 Rob 拿着邮件并且在 Sam 的办公室, 那么它就可以递邮件了。 dm 表示 Rob 递邮件。

假设 Rob 在一个时间内只能执行一个动作, 并且假定一个低层的控制器可以执行这些动作。

8.1.1 显式状态空间表示法

一种表示动作的结果和先决条件的方法是显式地列举出每一个状态, 以及每一状态下可能执行的动作, 对每一个状态-动作对, 列出在这一状态下执行这个动作后所到达的状态。这需要下面这样的一个表格:

状态	动作	结果状态
s_7	act_{47}	s_{94}
s_7	act_{14}	s_{83}
s_{94}	act_1	s_{23}
...

这个关系表中的第一行元组表示状态 s_7 时可能执行动作 act_{47} , 如果状态 s_7 下执行了动作 act_{47} , 那么结果状态将是 s_{94} 。

因此, 这就是以图的形式显式地表示动作, 称为状态空间图(state-space graph), 第 3 章用过此类图。

351

【例 8-2】 在例 8-1 中, 状态由一个五元组表示: 指定了机器人的位置, 机器人是否有咖啡, Sam 是否想要咖啡, 收发室是否有邮件, 机器人身上是否有邮件。例如, 元组

$\langle lab, \overline{rhc}, swc, \overline{mw}, rhm \rangle$

表示状态: Rob 在实验室, Rob 没有咖啡, Sam 想要咖啡, 收发室没有邮件, Rob 身上有邮件。

$\langle lab, rhc, swc, mw, rhm \rangle$

表示状态：Rob 在实验室，Rob 拿着咖啡，Sam 想要咖啡，收发室有邮件，Rob 身上没有邮件。

在这个例子中，总共有 $4 \times 2 \times 2 \times 2 \times 2 = 64$ 个状态。直观上，这些状态几乎都可以实现，即便你不相信一个智能机器人能够达到其中某些状态。

下面有 6 个动作，但不是每个状态下的所有这些动作都是可行的。

这些动作按照状态转移的形式定义如下：

状态	动作	结果状态
$\langle lab, rhc, swc, mw, rhm \rangle$	<i>mc</i>	$\langle mr, rhc, swc, mw, rhm \rangle$
$\langle lab, rhc, swc, mw, rhm \rangle$	<i>mcc</i>	$\langle off, rhc, swc, mw, rhm \rangle$
$\langle off, rhc, swc, mw, rhm \rangle$	<i>dm</i>	$\langle off, rhc, swc, mw, rhm \rangle$
$\langle off, rhc, swc, mw, rhm \rangle$	<i>mcc</i>	$\langle cs, rhc, swc, mw, rhm \rangle$
$\langle off, rhc, swc, mw, rhm \rangle$	<i>mc</i>	$\langle lab, rhc, swc, mw, rhm \rangle$
...

这个表格显示了两个状态之间的转换。完整的问题表示还包括其他 62 个状态之间的转换。

这种表示不是一种非常好的表示方法，原因有三：

- 通常有太多状态需要表示、获取和推理。
- 模型很小的改变就意味着表示上很大的改变。模型另一特征就是整体表示的改变。例如，要对机器人的电力层建模，使得机器人可以在实验室进行充电，那么所有的状态都要调整。
- 通常有大量的结构和行动效果的规则。这个结构可以使得对动作的先决条件、结果以及关于它们的推测的表达更加简洁和有效。

另一种方法是根据动作影响特征来对动作的结果建模。

352

8.1.2 基于特征的动作表示

基于特征的动作模型表示：

- 根据状态的特征值，判断一个状态下哪些动作是可能的；
- 当前状态和动作的特征值如何影响下一状态的特征值。

某一动作的**先决条件**是在执行这个动作前必须为真的命题。按照约束的形式，约束要求机器人只能选择先决条件为真的动作。

【例 8-3】 例 8-1 中，Rob 拿起咖啡(*puc*)的动作的先决条件是 $cs \wedge \overline{rhc}$ ，也就是说，Rob 必须在咖啡店(*cs*)，并且身上没有咖啡(\overline{rhc})。作为一个约束条件，意味着 Rob 在任何其他位置或者 *rhc* 为真时，动作 *puc* 都是不可行的。

顺时针移动的动作总是可能的，它的先决条件为真。

基于特征的动作表示(feature-based representation of actions)使用规则来为动作的结果状态的变量赋值。这些规则的实体可以包含执行的动作以及前一个状态的特征值。

规则有两种形式：

- **因果规则**(causal rule)指定一个特征获得一个新值的时间；
- **框架规则**(frame rule)指定一个特征保留其自身值的时间。

将这两种情况分开考虑是很有用的：什么情况使得该特征值改变，什么情况又使得该

特征值保持不变。

【例 8-4】 例 8-1 中, Rob 的位置取决于它先前的位置以及它向哪个方向移动。令 $RLoc'$ 表示指定结果状态位置的变量。下面的规则指定了 Rob 在咖啡店的一些条件。

$$RLoc' = cs \leftarrow RLoc = off \wedge Act = mcc$$

$$RLoc' = cs \leftarrow RLoc = mr \wedge Act = mc$$

$$RLoc' = cs \leftarrow RLoc = cs \wedge Act \neq mcc \wedge Act \neq mc$$

前两个规则是因果规则, 最后一个是框架规则。

机器人在结果状态中是否有咖啡取决于它前一个状态是否有咖啡以及它的动作:

$$rhc' \leftarrow rhc \wedge Act \neq dc$$

$$rhc' \leftarrow Act = puc$$

其中第一个是框架规则, 它指定机器人一直拿着咖啡直到送出咖啡为止, 这个规则隐含地告诉我们机器人不能丢弃或者遗失咖啡, 而且咖啡也不能被偷走。第二个是因果规则, 指定拿起咖啡的动作导致下一个时间步中机器人有了咖啡。

因果规则和框架规则并未指定一个动作可能在什么时间发生, 这是由动作的先决条件定义的。

8.1.3 STRIPS 表示法

前面的表示方法是基于特征的方法, 对每个特征都有规则来指定它的值, 即执行动作后到达的状态的特征值。另一种方法是基于动作的表示, 对每个动作指定它的结果。STRIPS 就是这样一个表示方法。STRIPS (STanford Research Institute Problem Solver), 即斯坦福研究所问题求解系统, 是用在机器人 Shakey 上的规划程序, Shakey 是第一个用人工智能技术设计的机器人。

首先, 将描述世界的特征分为**原始**(primitive)特征和**导出**(derived)特征。在任意给定状态下, 确定子句是用于从原始特征值决定导出特征值。STRIPS 表示方法用于确定某一状态下原始特征的值, 这一状态是基于先前的状态以及 Agent 所采取的动作而产生的。

STRIPS 表示法基于以下思想: 大多数的事情不受单一动作影响。对每一个动作, STRIPS 对何时该动作可能发生以及该动作会影响哪些原始特征进行建模。动作的结果依赖于 STRIPS 假设: 所有在动作的描述中没有提到的原始特征均保持不变。

某一动作的 STRIPS 表示由以下两部分组成:

- 先决条件是特征赋值的一个集合, 这些特征在这个动作发生时必须为真。
- 影响是执行动作后而发生改变的特征的结果赋值集合。

如果 $V=v$ 在动作 act 的结果列表上, 或者 act 的结果列表上没有提到特征 V , 并且在执行动作 act 之前 V 的值为 v , 那么执行动作 act 后原始特征 V 的值为 v 。每次的非原始特征值可以通过原始特征值导出。

当变量是布尔型时, 将结果分为**删除表**(delete list)(值为假的变量集合)和**添加表**(add list)(值为真的变量集合)。

【例 8-5】 例 8-1 中, Rob 拿起咖啡的动作(puc)有如下 STRIPS 表示:

先决条件: $[cs, \overline{rhc}]$

影响: $[rhc]$

也就是说, 机器人必须在咖啡店, 并且没有咖啡。该动作之后, 得到 rhc (即 $rhc = \text{true}$), 而其他的特征值没有受到该动作的影响。

【例 8-6】 递咖啡的动作(dc)可以定义为:

先决条件: $[off, rhc]$

影响: $\overline{rhc}, \overline{swc}$

机器人在办公室并且身上有咖啡时可以递咖啡。不管 Sam 想不想要咖啡它都可以递咖啡。如果在该动作之前 Sam 想要咖啡, 那么之后 Sam 就不想要了。因此, 动作的结果使得 $RHC = false$ 且 $SWC = false$ 。

基于特征的表示方法相对于 STRIPS 表示法更强大, 因为它可以表示 STRIPS 中所有可以表示的内容。但是却相对更冗长, 因为它需要显式的框架公理, 而这些公理是隐含在 STRIPS 表示中的。

某一动作集合的 STRIPS 表示可以转化成基于特征的表示, 如下所示。如果动作 act 的结果列表为 $[e_1, \dots, e_k]$, STRIPS 表示等价于因果规则:

$$e_i' \leftarrow act$$

这个动作使得所有 e_i 为真, 而框架规则:

$$c' \leftarrow c \wedge act$$

其中的每个条件 c 都不包含结果列表里的任一变量。这两个表示法中, 每个动作的先决条件都是一样的。

有条件的影响(condition effect)是动作的一个影响, 它依赖于其他特征的值。基于特征的表示法可以指定条件影响, 而 STRIPS 表示法却不能直接表示这些。

【例 8-7】 考虑动作 mc 的表示方法。 mc 的结果取决于执行动作前机器人的位置。

基于特征的表示如例 8-4 所示。

为了用 STRIPS 表示法表示动作 mc , 构造了多维动作(不同于先前初始值为真的动作)。例如, mc_{cs} (从咖啡店顺时针移动)的先决条件是 $RLoc = cs$, 结果是 $RLoc = off$ 。

8.1.4 初始状态和目标

在传统的规划问题中, 世界是完全可观察和确定的, 初始状态通过指定初始时刻每个特征的值来定义。

有两类目标:

- **实现目标(achievement goal):** 最终状态的命题必须为真。
- **维护目标(maintenance goal):** Agent 经过的每个状态必须为真命题。这些通常是**安全目标(safety goal)**——远离不好状态的目标。

也有一些其他类型的目标, 比如瞬时目标(在规划中的某处必须实现, 但是不需要维持到最后)或者资源目标(例如希望消耗最少的能量或者运行时间)等。

8.2 前向规划

确定性的规划(plan)是从一个给定的初始状态到达某一目标(goal)的动作序列。确定性的规划器(planner)是能够产生规划的问题求解程序。规划程序的输入是初始的世界描述、Agent 的可行动作说明以及目标描述。说明中包括动作的先决条件和影响。

其中一种最简单的规划策略是把规划问题看成是状态空间图中的一个路径规划问题。在**状态空间图**中, 节点表示状态, 弧表示从一个状态转移到另一个状态时所对应的动作。一个状态 s 的所有出弧对应这个状态下可以执行的所有的合法动作。也就是说, 对每个状态 s 以及状态 s 下先决条件为真的每个动作 a 都有一条对应的弧, 并且结果状态并不破坏维护目标。规划是从初始状态到达目标状态(满足实现目标的状态)的一条路径。

前向规划器(forward planner)是指从状态空间图的初始状态开始搜索直到到达一个目标状态(满足目标描述的状态)。它可以使用第 3 章介绍的任何一种搜索策略进行搜索。

【例 8-8】 图 8-2 表示的是搜索空间的一部分，其初始状态为：Rob 在咖啡店，Rob 没有咖啡，Sam 想要咖啡，收发室有邮件，Rob 身上没有邮件。不论目标状态如何，搜索空间都是一样的。

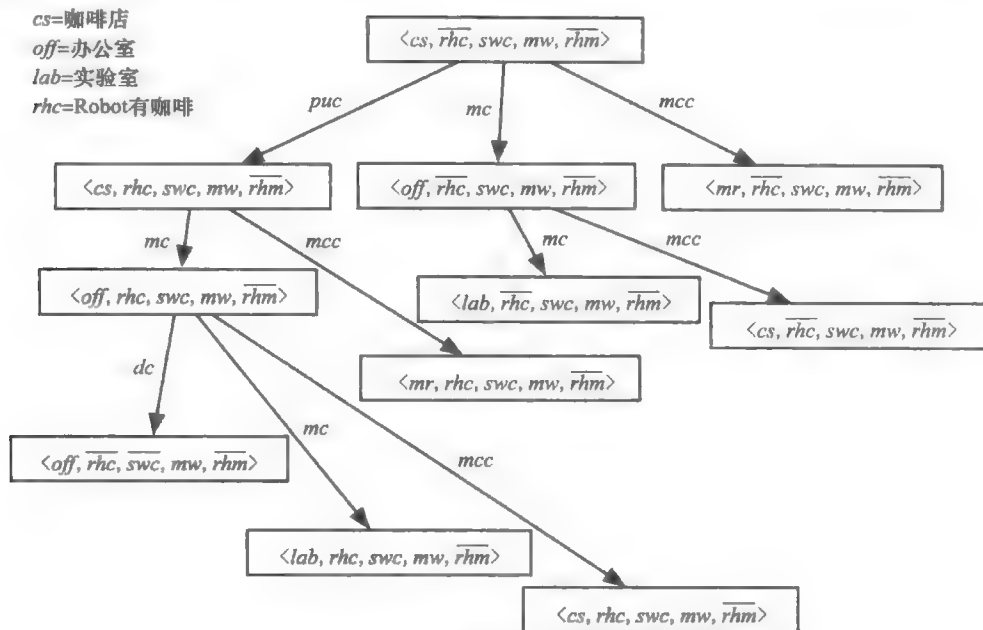


图 8-2 状态空间规划程序的部分搜索空间

使用前向规划程序不同于产生显式的基于状态的动作表示，因为图的相关部分可以从动作的表示中动态产生。

通过一个完整的搜索策略，例如带有多路径剪枝或者迭代深化搜索的 A^* 算法，一定能够找到一个解。搜索空间的复杂度由状态空间图的向前分支因子定义。分支因子是任一状态下所有可能动作的集合，可能非常大。对于简单的机器人传送领域，初始状态时分支因子是 3，其他情况下至多是 4。领域空间变大时，分支因子增大，这会导致搜索空间爆炸。找到某种好的启发式方法可能会降低复杂度（见习题 8.6），但是这个启发式算法必须足够好并且能够克服组合爆炸问题。

一个状态可以表示为：

(a) 完整的世界描述，即对每个原始命题的赋值或者作为定义状态的一个命题。

(b) 从某个初始状态出发的一条路径，即从初始状态到达该状态的一个动作序列。这样，未来的状态可以从指定动作结果的公理中推导出来。

为每个被创造的世界计算一个全新的世界描述，或者计算一个世界必要时的组成，这两者的区别相当于表达(a)与表达(b)间的区别。(b)使用的空间相对较小(特别是针对一个复杂的世界描述)，而且可以更快地创建一个新节点，但是在确定任一给定世界的未来情况时比较慢。(b)还有一个缺陷，就是在确定两个状态是否一致时比较耗时(例如，循环检测或多路径剪枝)。

作为一个前向搜索方法，我们已经介绍了状态空间搜索，但是也可以从满足目标的状态集合向后进行搜索。然而初始状态通常是完全可确定的，所以前沿可从单一的状态开始。但目标通常不能完全指定一个状态，以致将会有很多目标状态满足目标。这就意味着，前沿状态空间初始化时会非常大。因此，状态空间的向后搜索经常是不切实际的。

8.3 回归规划

在不同的搜索空间上进行搜索往往会取得更好的效果,在该搜索空间内节点不是状态而变成要实现的目标。一旦将这个问题转化为一个搜索问题,便可以使用第3章的任何一种算法。我们只考虑实现目标而不考虑维护目标(见习题8.9)。

356
↓
357

回归规划(regression planning)的搜索是在图中进行的,该图定义如下:

- 节点是一定要实现的目标。一个目标是(某些)特征的一个赋值集合。
- 弧相当于动作。特别的,从节点 g 到 g' 标记动作 act 的一条弧,表示 act 是实现目标 g 之前执行的最后一个动作,而节点 g' 是紧接着 act 之前必须为真的目标,以便 act 之后 g 为真。
- 开始节点是要实现的目标,假设它是特征的一个赋值集合。
- 如果初始状态 g 的所有元素的值为真,那么搜索的目标条件 $goal(g)$ 为真。

已知一个节点表示目标 g ,对于如下动作 act , g 的邻居节点是存在的:

- act 是可能的(possible): act 是可能会被执行的,并且 act 执行之后 g 为真。
- act 是有用的(useful): act 能实现部分 g 。

沿着带有动作 act 弧标记的目标 g 的邻居节点是 g' ,由最弱先决条件定义。在动作 act 之后目标 g 采用的最弱先决条件(weakest precondition)是一个这样的目标 g' :

- act 之前 g' 为真意味着紧接着 act 之后 g 为真。
- g' 是“最弱的”,即是任何满足第一条条件的命题必须隐含 g' 。这个排除了一些不必要的条件,如先决条件中一些不必要的条件。

如果对于给定的任一变量至多赋一个值,那么变量的一个赋值集合是相容的(consistent)。也就是说,如果给任何一个变量赋两个不同的值那么就会是不相容的。

假设考虑目标节点 $g = \{X_1 = v_1, \dots, X_n = v_n\}$ 。已知基于特征的动作表示方法,考虑计算一个节点的邻居节点。如果存在一个因果规则执行动作 act 获得 $X_i = v_i$,那么动作 act 是有用的。弧 act 上这个节点的邻居节点是命题:

$$precondition(act) \wedge body(X_1 = v_1, act) \wedge \dots \wedge body(X_n = v_n, act)$$

$body(X_i = v_i, act)$ 是一个规则实体中变量的赋值集合,这个规则指定动作 act 之后 $X_i = v_i$ 为真。如果对于某些 i 没有对应的规则,或者命题是不相容的(即对一个变量赋予不同的值),则没有该邻居。注意,如果同一动作有多个可行的规则,那么就会存在多个邻居。

358

对于 STRIPS 表示,对于某些 i 来说,如果 $X_i = v_i$ 是动作 act 的结果,则 act 对于解决 g 是非常有用的。除非对于 act 存在着结果 $X_j = v_j$,且 g 包含 $X_j = v'_j$,其中 $v'_j \neq v_j$,那么动作 act 是可能发生的。执行动作 act 之前, act 的先决条件和任何通过 act 不能达到的 $X_k = v_k$,都必须被保留。

这样,只要其为相容的,弧 act 上目标 g 的相邻节点为:

$$precondition(act) \cup (g \setminus effects(act))$$

【例8-9】 假设目标是实现 \overline{swc} ,开始节点是 $[\overline{swc}]$ 。如果在初始状态时就为真,则规划终止。否则,选择一个动作来实现 \overline{swc} 。这种情况下,只有动作 dc 满足条件。而 dc 的先决条件是 $off \wedge rhc$,因此有一条弧: $\langle [\overline{swc}], [off, rhc] \rangle$ 标记为 dc 。

考虑节点 $[off, rhc]$ 。有两个动作能够实现 off ,即 mc_cs 和 mcc_lab 。有一个动作能够实现 rhc ,即 puc 。然而, puc 有一个先决条件 $cs \wedge rhc$,但 cs 与 off 是不相容的(因为变量 $RLoc$ 有不同的赋值)。所以 puc 不是最后一个可能动作,即执行动作 puc 之后不可能实

现条件 $[off, rhc]$ 。

图 8-3 显示了前两层搜索空间(没有多路径剪枝或者循环检测)。可以注意到,无论初始状态是什么,搜索空间都是一样的。开始状态有两个角色,一个是停止条件,一个是启发式算法的一个源头。

下面这个例子展示了回归规划是如何确定最后一个动作的。

【例 8-10】 假设目标是 Sam 不想要咖啡,机器人身上有咖啡: $[swc, rhc]$ 。最后一个动作不可能是 dc 来实现 swc , 因为 dc 会实现 rhc 。最后一个动作一定是 puc 来实现 rhc 。所以结果目标是 $[swc, cs]$ 。同样,这个目标之前的最后一个动作也不能实现 swc , 因为它有一个先决条件 off , 而 off 和 cs 是不兼容的。因此,倒数第二个动作一定是一个移动动作以到达 cs 。

回归规划存在一个问题:某个目标可能是不可达的。确定某一目标集合是否是可达的通常难以从动作的定义中推断出来。例如,你应该知道一个物体不可能同时在两个不同的地点。有时候这些并不能明确的表示出来而只能隐含在动作的结果中,而事实上一个物体初始时只能在一个位置上。要实行一致性剪枝,回归规划程序可以使用领域知识对搜索空间进行修剪。

循环检测和多路径剪枝可以并入回归规划程序中。回归规划程序没有必要访问同一节点来修剪搜索。如果某一节点 n 表示的目标隐含通向 n 的路径上的一个目标,那么就可以去掉节点 n 。同样,多路径剪枝见习题 8.11。

即使没有特别的理由说明一个动作必须排在另一个动作之前,回归规划程序也必须是针对于某一特定的全序动作。如果动作之间相互影响很小,保证一个总排序势必会增加搜索空间的复杂性。例如,其可能会为了说明不存在成功的排序,而对一个动作序列检测其所有排序。

8.4 CSP 规划

在前向规划中,搜索受初始状态的限制,只能使用目标作为一个停止条件以及启发式方法的一个源。在回归规划中,搜索受目标的限制,只能将开始状态作为一个停止条件以及启发式方法的一个源。在同一规划中,我们可以向前或者向后搜索,通过使用初始状态去掉那些不可达的节点,使用目标去掉那些无用的节点。这个可以通过将一个规划问题转化成一个约束满足问题,然后使用第 4 章介绍的其中一个 CSP 方法来实现。

对于 CSP 表示,以特征的形式描述动作也是有用的,这里的特征即对动作和状态的一个抽象表示。表示动作的特征称为**动作特征**(action feature),表示状态的特征称为**状态特征**(state feature)。

【例 8-11】 对例 8-1 中的动作进行建模的另一方法是:每一步,Rob 可以选择

- 是否拿起咖啡。令 PUC 是一个布尔变量,机器人拿起咖啡时为真。
- 是否递送咖啡。令 $DelC$ 是一个布尔变量,机器人递送咖啡时为真。
- 是否取邮件。令 PUM 是一个布尔变量,机器人取邮件时为真。
- 是否递邮件。令 $DelM$ 是一个布尔变量,机器人递邮件时为真。
- 是否移动。 $Move$ 是定义域 $\{mc, mcc, nm\}$ 上的一个变量,依次表示 Rob 是顺时针移动、逆时针移动还是不动。

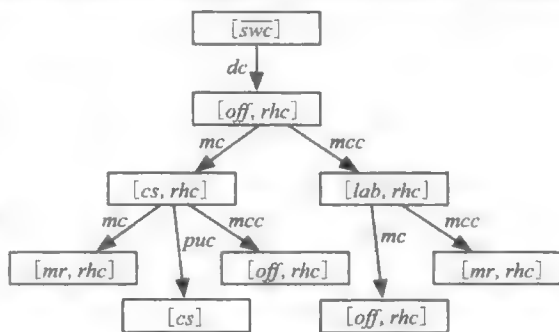


图 8-3 回归规划器的部分搜索空间

要将规划问题转化为一个 CSP 问题，首先需要选择一个固定的时域(horizon)，即规划执行的时间步数。假设这个值是 k ，CSP 有如下变量：

- 每一时刻每个状态特征对应一个变量，时间步从 0 到 k 。如果有 n 个特征，就有 $n(k+1)$ 个这样的变量。
- 每一时刻每个动作特征对应一个变量，时间步从 0 到 $k-1$ 。这些变量称为动作变量。 t 时刻的动作表示这个动作使得 Agent 从时刻 t 时的状态转移到时刻 $t+1$ 时的状态。

存在以下几种约束：

- **状态约束**(state constraint)，同一时间步内变量间的约束。这些可以包括对状态的自然约束或者可以确保禁止那些破坏维护目标的状态。
- **先决条件约束**(precondition constraint)，时刻 t 状态变量与动作变量之间的约束，即指定某一状态下哪些动作是可能的约束。
- **影响约束**(effect constraint)，时刻 t 的状态变量、时刻 t 的动作变量以及时刻 $t+1$ 的状态变量之间的约束，是根据 t 时刻的动作以及状态约束 $t+1$ 时刻状态变量的值的约束。
- **动作约束**(action constraint)，指定哪些动作不能同时发生。有时把这些称为互斥现象或者互斥约束(mutex constraint)。
- **初始状态约束**(initial-state constraint)，对初始状态的约束(时刻 0)。这些约束限制了初始状态是 Agent 的当前状态。如果只有一个初始状态，那么它可以表示成时刻 0 状态变量的一个域约束集合。
- **目标约束**(goal constraint)，限制了最终状态必须是一个能够满足实现目标的状态。这包括最后一步的目标是否是指定值，但这也可以是更一般的约束——例如，是否两个变量的值必须一样。

361

【例 8-12】 图 8-4 是使用 CSP 表示的某机器人传送系统，其中规划时域为 2。有三组状态变量：时刻 0 的初始状态；时刻 1 的状态；时刻 2 的最终状态。还有时刻 0 和时刻 1 的动作变量。

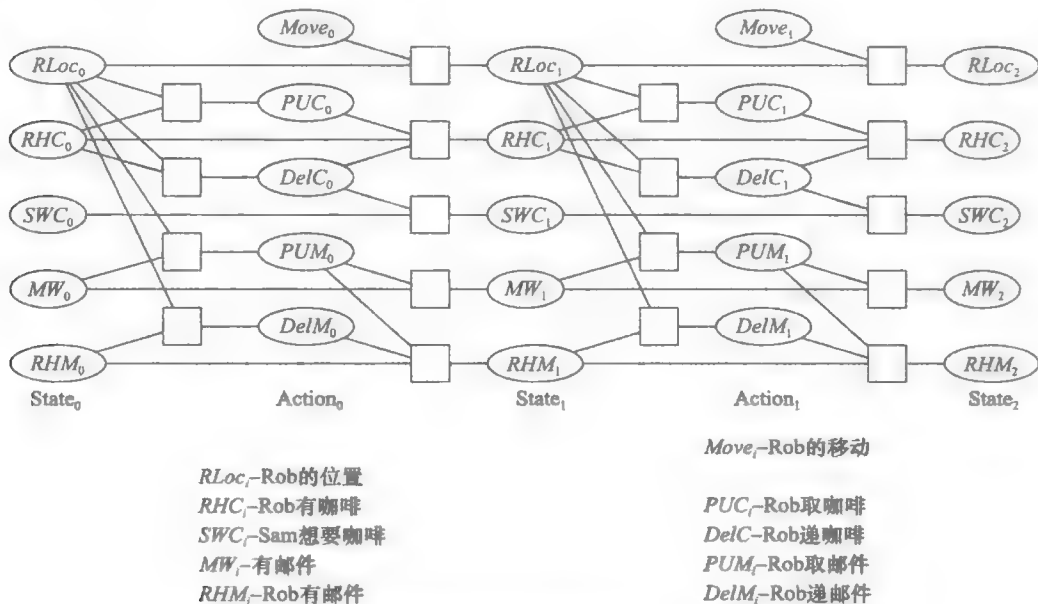


图 8-4 规划时域为 2 的传送机器人的 CSP 规划

在这个例子中没有域约束。若在这个领域以下内容为真，你可以声明一个约束：Rob 不能同时拿着咖啡和邮件，或者是当有邮件等着的时候 Rob 不能拿着邮件。这个例子中不包含这些。

动作左边的约束是先决条件约束，指定动作变量可以取什么值，即对每个动作变量来说什么动作是可行的。变量 $Move_i$ 没有先决条件，即所有的移动动作在任何状态下都是可行的。变量 PUM_i ，指定 Rob 是否可以取邮件，它取决于 i 时刻 Rob 的位置（即 $RLoc_i$ 的值）以及 i 时刻是否有邮件在等待（ MW_i ）。假设 Agent 可以选择不执行一个动作，那么一个动作的否定总是可行的（例如， $PUM_i = false$ ，写成 $\overline{pum_i}$ ）。只有当位置是 mr 并且 $MW_i = true$ 时，动作 $PUM_i = true$ ，即 pum_i 是可行的。当一个动作的先决条件是组合条件时，它可以写成一个约束集合，因为 CSP 中的约束是隐含相连的。如果先决条件是更复杂的命题，它可以表示为包括多于两个变量的约束。

362

时刻 1 以及时刻 2，状态变量左边的约束表示状态变量的值是前一个状态和动作的函数。例如，下面是 RHC_i 、 DC_i 、 PUC_i 以及下一状态机器人是否有咖啡 RHC_{i+1} 间的约束：

RHC_i	DC_i	PUC_i	RHC_{i+1}
true	true	true	true
true	true	false	false
true	false	true	true
true	false	false	true
false	true	true	true
false	true	false	false
false	false	true	true
false	false	false	false

这个表表示的约束与例 8-4 规则中的约束相同。

【例 8-13】 考虑寻找某规划使 Sam 可以拿到咖啡，规划时域为 2。

初始时，Sam 想要咖啡但是机器人没有咖啡，可以表示为两个域约束： SWC_0 和 RHC_0 。目标是 Sam 不再想要咖啡，可以表示为域约束 $SWC_2 = false$ 。

在这个网络上运行弧是一致性，其结果为 $RLoc_0 = cs$ （机器人必须在咖啡店开始）， $PUC_0 = true$ （机器人最初拿起咖啡）， $Move_0 = mc$ （机器人必须向办公室方向移动）， $DC_1 = true$ （机器人必须在时刻 1 送出咖啡）。

CSP 表示法假设规划时域是固定的（即一个固定的步数）。要找到任意步数的规划，这个算法可以运行 k 步（ $k=0, 1, 2$ ），直到找到解。对于随机局部搜索算法，通过搜索所有的时域（ k 从 0 到 n ），并且允许 n 慢慢地增加，有可能一次搜索到多个规划时域。当使用弧的一致性以及搜索求解 CSP 问题时，通过尝试更长的规划时域也可能不会获得更好的结果。也就是说，通过对为什么对于时域 n 内不存在解的分析，可以知道对于多于 n 步的范围内也不会存在任何规划。这使得在没有规划时会暂停规划程序。见习题 8.12。

8.5 偏序规划

前向和回归规划程序在规划过程的所有阶段实行的都是动作上的全序排序。而 CSP 规划程序确保执行动作的具体时间。这意味着，那些规划程序必须提交一组有序的动作，当把这些动作加入一个偏序规划时，这些动作是不能同时发生的，即使没有具体的原因把

一个动作放在另一个动作之前。

偏序规划(partial-order planner)程序的思想是动作之间有一个偏序排序,并且执行时只提交动作间的一个排序。有时也称为**非线性规划程序**(non-linear planner),其实这个说法用词并不准确,因为这些规划程序经常产生一个线性规划。

363

偏序关系是一种小于关系,它是传递的且不对称的。偏序规划是一个动作集合连同同一个偏序排序。这个偏序表示的是动作间的“之前(before)”关系。这样,任何一个符合偏序排序的动作的总排序集合,都能从初始状态求解目标状态。在偏序中如果动作 act_0 在 act_1 之前,就记为 $act_0 < act_1$ 。这意味着动作 act_0 必须发生在 act_1 之前。

为了统一起见,把 start 和 finish 均看成是一个动作。其中, start 初始状态时为真, finish 的先决条件是要求解的目标。伪动作 start 在所有其他动作之前, finish 在所有其他动作之后。这样,算法就不需要考虑初始状态和目标状态的特殊情形了。当 finish 的先决条件满足时,目标也就实现了。

一个既非 finish 又非 start 的动作,将会出现在一个偏序规划中来作为某一动作的先决条件。规划中动作的先决条件或者初始状态时为真,由 start 来完成,或者由规划中的某个动作来完成。

我们必须保证为动作所指定的先决条件可以完成。规划中动作 act_1 的每一个先决条件 P 都会有一个动作 act_0 与它相关,所以动作 act_0 实现 act_1 的先决条件 P 。表示为一个三元组 $\langle act_0, P, act_1 \rangle$,它是一个**因果关系**(causal link)。这个偏序关系指定动作 act_0 发生在 act_1 之前,记为 $act_0 < act_1$ 。任何其他的使得 P 为假的动作 A 要么发生在 act_0 之前,要么发生在 act_1 之后。

偏序规划的操作步骤大致如下:以动作 start、finish 以及偏序 $start < finish$ 开始。规划程序维持以下议程: $\langle P, A \rangle$ 对集合,其中, A 是规划中的一个动作, P 是必须实现的 A 的一个先决条件原子。初始时,议程包含对 $\langle G, finish \rangle$,其中 G 是在目标状态时必须为真的一个原子。

在规划过程的每个阶段,从议程中选出 $\langle G, act_1 \rangle$ 对,其中 P 是动作 act_1 的先决条件。然后选择一个动作 act_0 来实现 P 。这个动作要么已经在这个规划中,例如,有可能是动作 start;要么是即将添加到规划中的一个新动作。这个偏序中动作 act_0 必须发生在 act_1 之前。然后增加一个因果关系来记录: act_0 实现 act_1 的先决条件 P 。规划中删除 P 的任何动作必须发生在 act_0 之前或者 act_1 之后。如果 act_0 是一个新动作,那么就将其先决条件添加到这个规划中。程序持续进行直到议程为空。

这是一个非确定的过程,“choose(选择)”以及“either...or(或者……或者)”形成了必须搜索的选择,有两个选择需要搜索:

- 选择哪个动作来实现 G 。
- 删除 G 的动作是发生在 act_0 之前还是 act_1 之后。

图 8-5 给出了偏序规划算法。

364

函数 $add_const(act_0 < act_1, Constraints)$ 将约束 $act_0 < act_1$ 添加到 $Constraints$,并返回约束集。如果 $act_0 < act_1$ 与 $Constraints$ 不兼容,那么返回值为假。有许多方法可以执行这个函数(见习题 8.13)。

函数 $protect(\langle act_0, G, act_1 \rangle, A)$,检测是否 $A \neq act_0$,是否 $A \neq act_1$ 以及 A 是否删除 G 。如果是这样,要么添加 $A < act_0$,要么添加 $act_1 < A$ 到该约束集合中。这是一个已进行搜索的非确定性选择。

```

1: non-deterministic procedure PartialOrderPlanner(Gs)
2:   Inputs:
3:     Gs: 要完成的原子命题集合
4:   Output:
5:     为完成 Gs 的线性规划
6:   Local:
7:     Agenda:  $\langle P, A \rangle$  的集合,  $P$  是原子,  $A$  是动作
8:     Actions: 当前计划的动作集合
9:     Constraints: 动作上的暂时的约束集合
10:    CausalLinks:  $\langle act_0, P, act_1 \rangle$  元组的集合
11:    Agenda  $\Leftarrow \{ \langle G, finish \rangle : G \in Gs \}$ 
12:    Actions  $\Leftarrow \{ start, finish \}$ 
13:    Constraints  $\Leftarrow \{ start < finish \}$ 
14:    CausalLinks  $\Leftarrow \{ \}$ 
15:    repeat
16:      从 Agenda 中选择并且移除  $\langle G, act_1 \rangle$ 
17:      either
18:        选择  $act_0 \in Actions$ , 该  $act_0$  完成  $G$ 
19:      or
20:        选择  $act_0 \notin Actions$ , 该  $act_0$  完成  $G$ 
21:        Actions  $\Leftarrow Actions \cup \{ act_0 \}$ 
22:        Constraints  $\Leftarrow add\_const(start < act_0, Constraints)$ 
23:        for each  $CL \in CausalLinks$  do
24:          Constraints  $\Leftarrow protect(CL, act_0, Constraints)$ 
25:          Agenda  $\Leftarrow Agenda \cup \{ \langle P, act_0 \rangle : P \text{ 是 } act_0 \text{ 的先决条件} \}$ 
26:          Constraints  $\Leftarrow add\_const(act_0 < act_1, Constraints)$ 
27:          CausalLinks  $\Leftarrow CausalLinks \cup \{ \langle act_0, G, act_1 \rangle \}$ 
28:          for each  $A \in Actions$  do
29:            Constraints  $\Leftarrow protect(\langle act_0, G, act_1 \rangle, A, Constraints)$ 
30:    until Agenda =  $\{ \}$ 
31:    return 符合约束的动作的整体顺序

```

图 8-5 偏序规划算法

【例 8-14】考虑目标 $\overline{swc} \wedge \overline{mw}$, 初始状态为: $RLoc = lab, swc, \overline{rhc}, mw, \overline{rhm}$ 。
初始议程为:

$\langle \overline{swc}, finish \rangle, \langle \overline{mw}, finish \rangle$

假设选择 $\langle \overline{swc}, finish \rangle$ 从议程中删除。存在一个动作可以实现 \overline{swc} , 即送出咖啡 dc , 其先决条件是 off 和 rhc 。该循环(repeat loop)结束后, 议程包括:

$\langle off, dc \rangle, \langle rhc, dc \rangle, \langle \overline{mw}, finish \rangle$

约束是 $\{ start < finish, start < dc, dc < finish \}$ 。存在因果关系 $\langle dc, \overline{swc}, finish \rangle$, 这个因果关系意味着不允许某个发生在 dc 之后或者 $finish$ 之前的动作取消 \overline{swc} 。

假设从议程中选择 $\langle \overline{mw}, finish \rangle$ 。动作 pum 能够实现它, pum 的先决条件是 mw 和 $RLoc = mr$ 。将因果关系 $\langle pum, \overline{mw}, finish \rangle$ 添加到因果关系集合中, 并将 $\langle mw, pum \rangle$ 和 $\langle mr, pum \rangle$ 添加到议程中。

假设选择 $\langle mw, pum \rangle$, 动作 $start$ 可以实现 mw , 因为 mw 初始时为真。将因果关系 $\langle start, mw, pum \rangle$ 添加到因果关系集合中。议程中不做任何添加。

这个阶段, dc 与 pum 之间没有强制顺序关系。

假设从议程中删除 $\langle off, dc \rangle$ 。有两个动作可以实现 off : ms_cs , 先决条件是 cs ; mcc_lab , 先决条件是 lab 。算法搜索这些选项。如果选择了 mc_cs , 就添加因果关系 $\langle mc_cs, off, dc \rangle$ 。

当使用移动动作来实现 $\langle mr, pum \rangle$ 时,就会首次破坏因果关系。这个动作破坏了因果关系 $\langle mc_cs, off, dc \rangle$,它必须发生在 dc 之后或者 mc_cs 之前(机器人送完咖啡后再去收发室)。

前面的算法掩盖了一个重要的细节。一个规划中有时需要执行某个动作多次。这种情况下,前面的算法就不起作用了,因为它试图去找到发生在同一时刻的动作实例间的一个偏序关系。为了解决这个问题,排序应该是动作实例间而不是动作本身之间的关系。要实现这一点,我们为一个动作的每个实例分配一个索引,然后对这个动作实例的索引进行排序而不是动作本身。这个留作练习。

8.6 本章小结

- 规划是选择一个动作序列以实现某个目标的过程。
- 动作是一个状态到另一个状态的一个函数(状态转移函数)。一些表示方法都是从状态的表示中对其结构进行开发的。特别的,基于特征的动作表示法表示的就是在前一状态下必须为真使得某特征在下一个状态中获得一个值。STRIPS表示法是基于动作的表示,它指定动作的结果。
- 可以使用不同的规划算法将规划问题转化为搜索问题。

366

8.7 参考文献及进一步阅读

STRIPS表示法是由 Fikes 和 Nilsson[1971]提出的。

目前很多人在研究如何规划动作序列。Yang[1997]出版了一本教材,对规划进行了概述。一些经典的论文可以参看 Allen、Hendler 和 Tate[1990]。

前向规划已经成功地用于积木世界规划中,其中一些好的启发式方法是由 Bacchus 和 Kabanza[1996]定义的。(见习题 8.6。)

回归规划最早由 Waldinger[1977]提出。而最弱先决条件的使用是基于 Dijkstra[1976]的工作,用它来定义命令式编程语言的语义。这应该不会太奇怪,因为命令式语言的命令是改变计算机状态的动作。

CSP 规划是基于图规划[Blum 和 Furst, 1997]和表规划[Kautz 和 Selman, 1996]的。Lopez 和 Bacchus[2003]以及 van Beek 和 Chen[1999]也曾研究过将规划看成是一个 CSP 问题。Bryce 和 Kambhampati[2007]给出了近期的研究现状。

偏序规划首次出现在 Sacerdoti[1975]的 NOAH 系统中,随后又出现在 Tate[1977]的非线性(NONLIN)系统,Chapman[1987]的调整(TWEAK)算法,以及 McAllester 和 Rosenblitt[1991]的系统非线性规划(SNLP)算法中。参见 Weld[1994]对偏序规划的综述以及 Kambhampati, Knoblock 和 Yang[1995]对这些算法的比较。这里给出的版本主要是 SNLP(见习题 8.15)。

参见 Wilkins[1988]对规划中实际问题的探讨,以及 Weld[1999], McDermott 和 Hendler[1995], Nau[2007]的相关文章对规划问题的最新综述。

8.8 习题

8.1 考虑图 8-1 的规划领域。

- 给出基于特征表示的特征 MW 和 RHM。
- 给出动作取邮件和送邮件的 STRIPS 表示。

8.2 假设机器人不能同时拿着咖啡和邮件。给出两种方法通过改变表示这个规划问题的 CSP 来反映这个约束。通过给出一个问题对其进行验证,考虑当机器人存在该约束与不存在该约束时问题结果的异同。

8.3 写出这个有限机器人传送世界的一个完整描述,然后画出一个状态空间表示图,其中至少包括本章讨论的积木世界中的两个动作实例。注意,弧的数量取决于动作实例的个数。

367

8.4 改变传送机器人世界的表示(例 8-1),使得

(a) Agent 不能同时拿着邮件和咖啡。

(b) Agent 可以拿着一个盒子,这个盒子可以放置物体(这样,Agent 就可以拿着盒子,而盒子可以同时装着邮件和咖啡)。

在一个例子上进行测试,给出一个不同于原始表示的解。

8.5 假设我们必须求解打扫房子的规划问题。不同的房间有的可以擦(使房间无尘),有的可以扫(使房间有一个干净的地板),但如果机器人在某个房间就只能擦或者扫。扫地会使得房间落满灰尘(即不是无尘)。如果抹布是干净的,机器人只可以擦一个房间。但擦房间会造成额外的灰尘,例如擦车库会使抹布变脏。机器人可以直接从一个房间移动到另一个房间。

假设有两个房间:仓库,如果它是积满灰尘的,会造成额外的灰尘(extra dusty);卧室,非额外灰尘。有以下特征:

- 当卧室落满灰尘时, Lr_dusty 为真。
- 当仓库落满灰尘时, Gar_dusty 为真。
- 当卧室地板脏时, Lr_dirty_floor 为真。
- 当仓库地板脏时, Gar_dirty_floor 为真。
- 当抹布干净时, Dustcloth_clean 为真。
- Rob_loc 表示机器人的位置。

假设机器人任何时刻都可以做以下的某个动作:

- move: 移动到其他房间。
- dust_lr: 擦卧室(机器人在卧室且卧室落满灰尘)。
- dust_gar: 擦仓库(机器人在仓库且仓库落满灰尘)。
- sweep_lr: 扫卧室地板(机器人在卧室)。
- sweep_gar: 扫仓库地板(机器人在仓库)。

(a) 给出 dust_gar 的 STRIPS 表示。

(b) 给出 lr_dusty 的基于特征表示。

(c) 假设,动作 sweep 代替动作 sweep_lr 和 sweep_gar,意思是无论机器人在哪个房间只是清扫。

解释如何改变先前的解答来处理这个新的表示或者为什么不能使用这个新表示。

8.6 针对前向规划程序,提出一个好的启发式方法应用在机器人传送领域,实现它,运行效果如何?

8.7 假设你对动作 a_1 和 a_2 有 STRIPS 表示,你想要定义混合动作 a_1, a_2 的 STRIPS 表示,也就是先执行动作 a_1 ,再执行动作 a_2 。

(a) 这个混合动作的结果列表是什么?

(b) 这个混合动作的先决条件是什么?假设先决条件可以表示为一个 Variable=value 对列表(而不是任意的逻辑公式)。

(c) 使用例 8-1 的传送机器人领域,给出混合动作 puc、mc 的 STRIPS 表示。

(d) 给出混合动作 puc、mc、dc(由 3 个原始动作组成)的 STRIPS 表示。

(e) 给出混合动作 mcc、puc、mc、dc(由 4 个原始动作组成)的 STRIPS 表示。

8.8 在前向规划中,可以用到达哪个状态的动作序列来表示一个状态。

(a) 解释如何检查一个动作的先决条件是否满足并给出一个这样的表示。

(b) 解释在这样一个表示中如何执行循环检测。你可以假设所有的状态都是合法的。(其他的程序已经确保满足了先决条件。)

(提示:考虑习题 8.7 中的混合动作,由任一阶段的前 k 个或者倒数 k 个动作组成。)

8.9 解释对于不论是动作的基于特征表示还是 STRIPS 表示,如何将回归规划程序扩展到包括维护目标。(提示:考虑当维护目标提到一个没有出现在节点中的特征时会发生什么。)

8.10 在传送机器人领域,为回归规划程序给出一个非零的启发式函数和一个实际路径成本的较低评估,是否可行?

8.11 解释如何将多路径剪枝并入回归规划程序中。应该何时剪掉节点?

8.12 对于 CSP 规划给出这样一种情况：当弧的搜索一致性在某一规划时域下不满足时，意味着不会有长于该时域的解了。（提示：考虑一个非常长的时域，前向搜索和后向搜索相互不受影响。）实现它。

8.13 要执行偏序规划程序中的函数 $add_constraint(A_0 < A_1, Constraints)$ ，必须为一个偏序选择一个表示。以下是一个偏序的不同表示，执行它们：

(a) 将一个偏序表示为推导排序的一个小于关系集合，例如，表示为列表 $[1 < 2, 2 < 4, 1 < 3, 3 < 4, 4 < 5]$ 。

369

(b) 将一个偏序表示为由排序推导出的一个小于关系集合，例如，表示为列表 $[1 < 2, 2 < 4, 1 < 4, 1 < 3, 3 < 4, 1 < 5, 2 < 5, 3 < 5, 4 < 5]$ 。

(c) 表示为一个 $\langle E, L \rangle$ 对集合，其中 E 是偏序中的一个元素， L 是偏序中元素 E 之后的所有元素的一个列表。对每个 E ，存在唯一的形式 $\langle E, L \rangle$ 。这样的例子是， $\langle 1, [2, 3, 4, 5] \rangle$ ， $\langle 2, [4, 5] \rangle$ ， $\langle 3, [4, 5] \rangle$ ， $\langle 4, [5] \rangle$ ， $\langle 5, [] \rangle$ 。

对于以上的每个表示，这个偏序能有多大？检查一个新的排序的一致性有多容易？添加一个新的小于排序约束有多容易？你认为哪个是最高效的表示？你能想出一个更好的表示吗？

8.14 用于偏序规划程序中的选择算法并不十分复杂。对选择的子目标进行排序可能是一种明智的选择。例如，在机器人世界，机器人应该尽量在子目标 at 之前实现 $carrying$ 子目标。因为对机器人来说，当它知道应该拿起一个物体时尽快地拿起它可能是明智的。然而，机器人并不一定想移动到某个地方，除非它正携带着需要携带着的一切。实现这样的启发式选择算法。选择启发式是否实际上比选择最后添加的子目标更好？你能否想到一个一般的选择算法，不需要知识工程师对每一对子目标进行排序？

8.15 SNLP 算法与这里介绍的偏序规划程序是一样的，只是在 $protect$ 程序中，先决条件是：

$A \neq A_0$ 且 $A \neq A_1$ 且 (A 删除 G 或者 A 实现 G)

这个要求系统性，系统性意味着对每个线性规划有唯一的偏序规划。解释为什么系统性可能是也可能不是一件好事（例如，讨论它是如何更改分支因子或减小搜索空间的）。在不同的例子上测试不同的算法。

370

不确定性规划

规划就像在一个建筑的周围搭建的脚手架。当你建造建筑外壳时，脚手架的作用至关重要；一旦建好了建筑外壳并开始内部装修，脚手架就需要拆掉了。这就是我所理解的规划。为顺利完成一项工作，规划工作必须经过深思熟虑，但它不能代替你在完成工作中的每个细节时付出的努力。实现你想法的过程很少会与你的原有规划一致。

——Twyla Tharp[2003]

在上面的引文中，Tharp 指的是舞蹈，但对于任何一个具有不确定性的 Agent 来说，该观点同样适用。一个 Agent 不能仅仅规划出一个顺序的步骤，规划的结果需要更加复杂。做规划时，必须考虑到这种情况：Agent 并不知道在它采取行动时会发生什么。Agent 应该规划如何应对其所处环境的变化。Agent 的具体行为应由事先的规划和其遇到的真实环境决定。

如果一个 Agent 不能够确切地知道它的行动能够产生什么影响，那么请考虑一下它该怎么做？决定该怎么做是很困难的一件事，因为在任何时候，一个 Agent 应该做什么需要依赖于它在将来要做什么。然而，它将来要做出的行为又依赖于它现在的动作及它在将来观察到的东西。

在有不确定性的情况下，一个 Agent 通常不能够保证满足其目标，甚至试图以最大的可能达到其目标也可能变得不切实际。例如，一个目标为不在车祸中受伤的 Agent 可能会拒绝坐进汽车或沿着人行道走路，甚至不会进入建筑物的第一层——大多数人都会认为这样的 Agent 不是很聪明。一个不能保证满足其目标的 Agent 的失败方式多种多样，其中一些相当糟糕。

本章讨论的是如何同时考虑这些因素。

一个 Agent 决定要干什么依赖于三件事情：

- Agent 的能力：Agent 必须从其可用选项中选出一个。
- Agent 的信念：你可能会试着解释“这个世界上什么为真”。对于 Agent 则不一样：当它不知道世界上什么为真时，它就只基于自己的信念采取动作。Agent 根据其对外界的感知而更新自己的信念。
- Agent 的偏好：当一个 Agent 必须在不确定环境下进行推理时，它不仅需要考虑最可能会发生什么，而且需要考虑会发生什么。某些有可能发生的事情可能会造成很严重的后果。这里的术语“目标”的含义较第8章中的更丰富，因为 Agent 的设计者必须明确不同结果间的平衡关系。例如，一些行为经常会产生很好的结果，但有时也会导致灾难性的后果，那么它就必须比较决定是否选择另一个可选的动作：这个动作能够以较小的概率产生好的结果，通常只能得到一般的结果，但会以较小的概率导致灾难的结果。决策理论用来解决平衡产生期望的结果与其他结果的问题。

谁来评价?

任何一个计算机程序或任何一个人在决定采取什么行动或给出什么建议时, 都会使用某种评价系统来判定什么是重要的, 什么不是重要的。

Alice... 继续问: “你能告诉我应该选择哪条路吗?”

“这取决于你想去哪里。”Cat 答道。

“我不在意我会走到哪里。”Alice 说。

“那么你选哪条路都无关紧要。”Cat 答道。

——Lewis Carroll(1832—1898), 《Alice's Adventures in Wonderland》, 1865

很自然的, 我们所有人都希望计算机按照我们的评价系统来工作, 但它们却不能按照任何一个人的评价系统工作! 如果你开发一个在实验室工作的程序, 那么没有问题, 它会按照人的希望工作。此时程序会根据设计者的目标和评价标准执行动作, 同时设计者也是程序的使用者。如果一个系统有多个用户, 那你就必须考虑将谁的评价标准植入程序中。如果一个公司向一个医生销售一套医疗诊断系统, 那么该系统给出的诊断能够反映社会、公司、医生、病人(他们可能有非常不同的评价标准)的评价标准吗? 它能够测定医生或者病人的评价标准吗?

如果你要开发一个能给别人提供一些建议的系统, 你就需要找出哪些是符合预期的, 同时也要找出他们的评价标准是什么。例如, 在一个医疗诊断系统中, 正确的诊断过程不仅依赖于病人的症状, 而且依赖于病人优先考虑的事情。为了更好地了解其所处的环境, 他们准备忍受一些痛苦吗? 他们愿意为了延长一点儿生命而忍受许多疼痛吗? 他们准备承受什么样的风险? 对于一个建议你做什么事情的程序或人, 如果他没有询问你的意见, 则你需要保留怀疑态度。作为能执行任务或提供建议的程序的开发者, 你应该清楚地知道要在程序中嵌入谁的评价系统。

9.1 偏好和效用

一个 Agent 应该根据其偏好来决定做什么。在本节, 我们介绍几个我们希望的偏好的直观性质, 并给出由这些性质得到的一个结果。我们给出的属性是**理性公理**(axioms of rationality), 从它们出发, 我们证明一个有关如何测度偏好的定理。你需要考虑这些公理对于一个**理性 Agent**而言是否是合理的; 如果你认为它们是合理的, 你就应该接受由它们导出的结果。如果你不接受这个结果, 你就应该考虑需要放弃哪一条公理。

一个 Agent 根据其各个行动能产生的**结果**(outcome)来选择其行动。Agent 所选行动产生的结果是其偏好的结果。如果 Agent 没有特别的偏好, 则 Agent 做什么都无关紧要。我们在考虑行动产生的结果时先不考虑相关联的行动所产生的结果, 并假设这些结果的数量是有限的。

我们在结果上定义了偏好关系。假设 o_1 和 o_2 是两个结果。如果 o_1 的合意度不比 o_2 低, 则我们称 o_1 **弱偏好**(weakly preferred)于 o_2 , 记为 $o_1 \geq o_2$ 。下面的公理可以说是这个偏好关系的合理的性质。

定义 $o_1 \sim o_2$ 来表示同时满足 $o_1 \geq o_2$ 和 $o_2 \geq o_1$ 。也就是说, $o_1 \sim o_2$ 表示这两个结果的合意度相同。此时, 我们说 Agent 在结果 o_1 与 o_2 之间**没有偏好**。

定义 $o_1 > o_2$ 来表示 $o_1 \geq o_2$ 且 $o_2 \not\geq o_1$ 。也就是说, Agent 偏好结果 o_1 胜过结果 o_2 , 且此差异有显著性。此时, 我们称 o_1 **强偏好**(strictly preferred)于 o_2 。

一般来说, 一个 Agent 事先不能确切地知道其行为产生的结果。通过在各个结果上分配有限的概率来定义一个抽奖法(lottery), 记为

$$[p_1: o_1, p_2: o_2, \dots, p_k: o_k]$$

其中, o_i 是 Agent 的行为产生的各种结果, p_i 是非负实数且满足

$$\sum_i p_i = 1$$

上述抽奖法规定了结果 o_i 的发生概率是 p_i 。在后文叙述中, 假设所有结果都包含抽奖法, 对于抽奖法来说也同样包含有抽奖法。

公理 9.1(完备性) Agent 在所有结果上都定义了偏好:

$$\forall o_1 \forall o_2 \quad o_1 \geq o_2 \text{ 或 } o_2 \geq o_1$$

公理 9.1 的合理性在于一个 Agent 必须采取行动。如果 Agent 执行了可用的行为能够产生结果 o_1 和 o_2 , 那么它就会显式或隐式地更希望得到其中的某个结果, 而不希望得到另一个结果。

公理 9.2(可传递性) 偏好一定是可传递的:

$$\text{如果 } o_1 \geq o_2 \text{ 并且 } o_2 \geq o_3, \text{ 那么有 } o_1 \geq o_3.$$

为说明公理 9.2 的合理性, 首先假设该公理是错误的, 即当 $o_1 \geq o_2$ 并且 $o_2 \geq o_3$ 时有 $o_3 > o_1$ 。因为 o_3 强偏好于 o_1 , 所以 Agent 会乐意付出一些代价将 o_1 结果转换成 o_3 。假设 Agent 得到了 o_3 结果, 由于结果 o_2 至少与结果 o_3 一样好, 所以 Agent 会乐意得到结果 o_2 。进一步, 结果 o_1 至少与结果 o_2 一样好, 所以 Agent 会乐意得到结果 o_1 。一旦 Agent 得到结果 o_1 , 它就会乐意付出一些代价从而得到结果 o_3 。此时, Agent 通过偏好关系走了一个循环, 付出了成本又回到了原地。这个含有成本付出的循环被称为货币泵, 因为只要循环足够多次, Agent 必须付出的成本会超过任何有限的数值。因此, 在一组结果之中循环并为之付出成本是不符合理性的, 从而, 一个理性的 Agent 对结果的偏好应该存在可传递性。

此外, 假设混合的 $>$ 和 \geq 关系满足单调性, 因此, 如果公理 9.2 的前提中的一个或两个偏好关系是强偏好的, 那么其结论也是强偏好的。也就是说, 如果 $o_1 > o_2$ 并且 $o_2 \geq o_3$, 那么有 $o_1 > o_3$; 或者, 如果 $o_1 \geq o_2$ 并且 $o_2 > o_3$, 那么有 $o_1 > o_3$ 。

公理 9.3(单调性) 一个 Agent 偏向于以较大的概率(而不是以较小的概率)得到较好的结果, 即如果有 $o_1 > o_2$ 且 $p > q$, 则有

$$[p: o_1, (1-p): o_2] > [q: o_1, (1-q): o_2]$$

注意, 在公理 9.3 中, 结果之间的符号 $>$ 表示 Agent 的偏好, 而 p 与 q 之间的 $>$ 表示两个数值间的大小比较。

公理 9.4(可分解性)[赌博中无乐趣] 对于两个抽奖法, 如果对于相同的结果其产生的概率是相同的, 则 Agent 对这两个抽奖法没有偏好, 对于抽奖法嵌套抽奖法的情况也是如此。例如:

$$[p: o_1, (1-p): [q: o_2, (1-q): o_3]] \sim [p: o_1, (1-p)q: o_2, (1-p)(1-q): o_3]$$

另外, 对于任意的结果 o_1 和 o_2 有 $o_1 \sim [1: o_1, 0: o_2]$ 。

公理 9.4 说明抽奖法只与其能够产生的结果及产生结果的概率有关。如果一个 Agent 有赌博的偏好, 那么赌博应成为结果空间的一部分。

上述公理能用来刻画一个 Agent 关于结果与抽奖法的许多偏好。假设有 $o_1 > o_2$, $o_2 > o_3$, 考虑对于不同的概率 p , $p \in [0, 1]$, 该 Agent 是否偏向获得如下结果:

• o_2

• 抽奖法 $[p: o_1, (1-p): o_3]$

当 $p=1$ 时, Agent 偏向于上述抽奖法(因为该抽奖法等价于 o_1 , 并且有 $o_1 > o_2$)。当

$p=0$ 时, Agent 偏向于 o_2 (因为此时上述抽奖法等价于 o_3 , 并且 $o_2 > o_3$)。随着 p 的变动, 在某一取值处, Agent 的偏好在 o_2 与上述抽奖法之间突然发生了变动。图 9-1 说明了偏好随 p 的取值变化而发生变动的情况, 其中 X 轴表示 p 的取值, Y 轴表示偏好的结果。

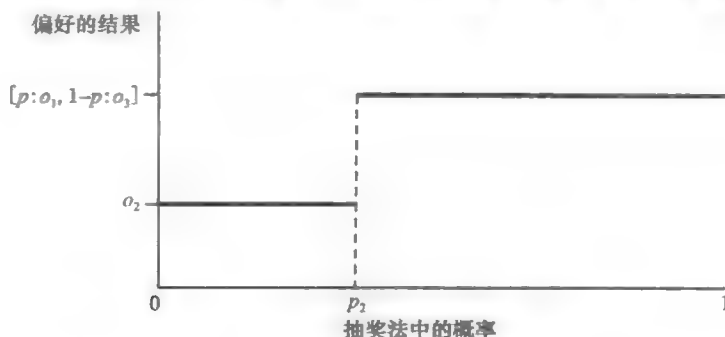


图 9-1 o_2 与抽奖法间的偏好是 p 的一个函数

命题 9.1 如果一个 Agent 的偏好关系是完备的、可传递的, 满足单调性和可分解性公理, 则如果有 $o_1 > o_2$, $o_2 > o_3$, 那么存在一个 $p_2 (0 \leq p_2 \leq 1)$ 满足:

- 对于所有的 $p < p_2$, Agent 偏向于 o_2 , 而不是上述抽奖法 (即 $o_2 > [p:o_1, (1-p):o_3]$)。
- 对于所有的 $p > p_2$, Agent 偏向于上述抽奖法 (即 $[p:o_1, (1-p):o_3] > o_2$)。

证明: 根据单调性和可传递性, 对于任意的 p , 如果 $o_2 \geq [p:o_1, (1-p):o_3]$, 则对于任意的 $p' < p$ 有 $o_2 > [p':o_1, (1-p'):o_3]$ 。相似地, 对于任意的 p , 如果 $[p:o_1, (1-p):o_3] \geq o_2$, 则对于任意的 $p' > p$ 有 $[p':o_1, (1-p'):o_3] > o_2$ 。根据完备性, 对于任意的 p , 只存在三种情况: $o_2 > [p:o_1, (1-p):o_3]$, $o_2 \sim [p:o_1, (1-p):o_3]$, $[p:o_1, (1-p):o_3] > o_2$ 。如果存在一个 p 满足 $o_2 \sim [p:o_1, (1-p):o_3]$, 则命题得证。对于满足 Agent 要么偏好 o_2 要么偏好 $[p:o_1, (1-p):o_3]$ 的某一个 p , 均对于所有大于 p 或小于 p 的值蕴含着相同的偏好。不断切分那些我们还不知其偏好的区域, 我们会得到一个极限值, 该值就是满足命题的 p_2 。 ■

命题 9.1 没有明确 Agent 在 p_2 时的偏好。下面这个公理说明 Agent 在这个取值上没有偏好。

公理 9.5 (连续性) 假设有 $o_1 > o_2$, $o_2 > o_3$, 则存在一个 $p_2 \in [0, 1]$ 使得

$$o_2 \sim [p_2:o_1, (1-p_2):o_3]$$

下面一个公理表明, 如果把一个抽奖法中的结果替换成另一个不会更差的结果, 则抽奖法不会变得更差。

公理 9.6 (可替换性) 如果 $o_1 \geq o_2$, 那么 Agent 会弱偏好将 o_2 替换为 o_1 而其他都相同的抽奖法。也就是说, 对于任意的 p 和结果 o_3 , 有:

$$[p:o_1, (1-p):o_3] \geq [p:o_2, (1-p):o_3]$$

该公理的一个直接的推论是可以将一个结果替换为另一个 Agent 没有偏好的结果而不改变其偏好。

命题 9.2 如果一个 Agent 满足可替换性公理且 $o_1 \sim o_2$, 那么 Agent 对那些仅是结果 o_1 和 o_2 不同的抽奖法无偏好。也就是说, 对于任意的 p 和结果 o_3 , 下面的无偏好关系成立:

$$[p:o_1, (1-p):o_3] \sim [p:o_2, (1-p):o_3]$$

该命题成立是因为 $o_1 \sim o_2$ 等价于 $o_1 \geq o_2$ 且 $o_2 \geq o_1$ 。

如果一个 Agent 满足完备性、可传递性、单调性、可分解性、连续性和可替换性公理，那么就定义该 Agent 是理性的(rational)。

这个关于理性的技术定义是否符合你关于理性的直觉概念由你来作出判断。在本节的后面部分，我们给出由此定义得出的结果。

偏好关系看起来可能很复杂，不过，下面的定理表明一个理性的 Agent 赋予一个结果的价值能够用一个实数来表示，而这些实数可以用来表示概率，因此，不确定环境下的偏好可以通过期望进行对比。这有点令人意外，原因是：

- 乍看起来，偏好反映的因素过多，因此难以用一个数值表示。例如，尽管一个人可能试图使用美元的多少来衡量偏好，但并不是所有的东西都有一个用来销售的价格或者难以将其价值转换成美元或美分。
- 你可能不认为那个数值能与概率联系起来。对于任意的货币值 x 和 y 及 $p \in [0, 1]$ ，一个对于价值为 $\$(px + (1-p)y)$ 和按抽奖法 $[p: \$x, (1-p) \$y]$ 确定的货币价值并无偏好的 Agent 被称为期望货币价值(expected monetary value, EMV) Agent。大多数人都不是 EMV Agent，因为他们在诸如 $\$1\,000\,000$ 与根据抽奖法 $[0.5: \$0, 0.5: \$2\,000\,000]$ 确定的美元数之间有很强的预设偏好(想象一下，你是愿意直接得到一百万美元呢？还是愿意执一枚硬币，当出现正面时什么也得不到，而当出现反面时得到两百万美元？)。钱的多少不能简单地与概率联系起来，因此用一个数值表示偏好的概率自然令人感到惊奇。

命题 9.3 如果一个 Agent 是理性的，则对于所有的结果 o_i 都有一个实数 $u(o_i)$ ，称为 o_i 的效用(utility)，满足

- $o_i > o_j$ ，当且仅当 $u(o_i) > u(o_j)$ 。

- 效用与概率呈线性关系

$$u([p_1: o_1, p_2: o_2, \dots, p_k: o_k]) = p_1 u(o_1) + p_2 u(o_2) + \dots + p_k u(o_k)$$

证明：如果 Agent 没有强偏好(即 Agent 对于得到的任何结果都一视同仁)，则对于所有的结果 o 定义其效用为 $u(o) = 0$ 。

另一方面，选择最好的结果记为 o_{best} ，选择最差的结果记为 o_{worst} ，并对于任意的结果 o ，定义其效用为 p ，其中 p 满足：

$$o \sim [p: o_{best}, (1-p): o_{worst}]$$

命题的前半部分满足可替换性和单调性。

命题的后半部分能够通过将 o_i 替换为等价的、介于 o_{best} 与 o_{worst} 之间的抽奖法，以及公理中给出的效用得到证明。证明细节留作练习。 ■

在上述证明中，效用全部介于 $[0, 1]$ 之间，但任意的线性刻度都能得到同一结论。有时候， $[0, 100]$ 是一个很好的刻度，这样可以使它和概率明确区分开来；在另一些时候，对于需要付出成本才能得到的结果来说，负数也是有用的。通常，一个程序应该接受任意的对于用户来说直观的刻度。

在钱数与效用之间，并不常常存在线性关系，即使每一结果都附有货币价值时也是如此。当涉及钱时，人们常常是风险规避(risk averse)的。他们宁愿稳稳地拿在手中 n 美元，而不愿参与一个获得钱数的期望为 n 的随机过程(在这个过程中可能得到更多回报，也可能得到较少回报)。

【例 9-1】 图 9-2 显示了一个风险规避 Agent 的金钱-效用平衡的曲线。风险规避对应一个凹效用函数。

这个 Agent 可能宁愿获得 30 万美元，而不愿以 50% 的概率获得 100 万美元 50% 的概率

什么也得不到，但是他又偏向于豪赌 100 万美元而不是直接得到 27 万 5 千美元。他愿意选择以 73% 的概率博取 100 万美元的赌局，而不愿选择以 50% 的概率博取 50 万美元的赌局。

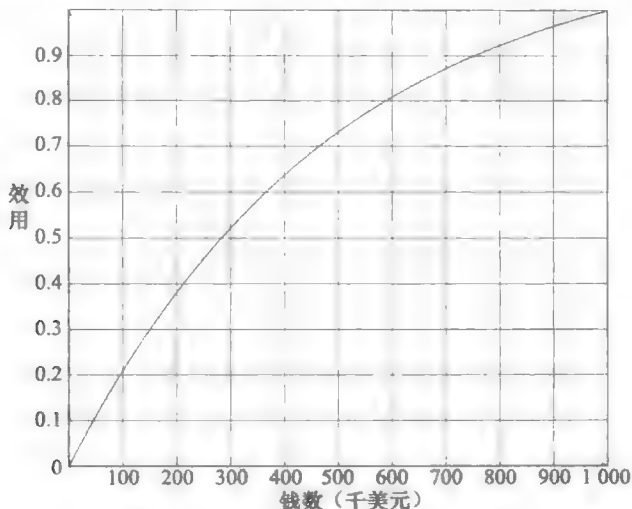


图 9-2 一个风险规避智能体的金钱-效用平衡图

注意，效用函数 $u(\$999\,000) \approx 0.9997$ 。因此，对于这个效用函数，人们可能会愿意付 1 000 美元来消除造成失去所有钱的 0.03% 的概率。这就是保险(insurance)公司存在的原因。通过付给保险(insurance)公司一些钱，比如 600 美元，Agent 能够将抽奖法的价值由 99 万 9 千美元增加到 100 万美元，而保险公司期望的赔付是 300 美元，期望的盈利是 300 美元。保险公司能够通过为足够多的房产提供保险来获得其期望的收益。这对各方都是有益的。

正如前面所述，理性与效用函数的形式没有关系。

【例 9-2】图 9-3 显示了一个人的金钱-效用平衡曲线，该人十分想要一个价值 30 美元的玩具，但是他也乐意要一个价值 20 美元的玩具。除此之外，钱对于 Agent 的决策影响不大。该 Agent 还准备承担风险来得到他想要的。比如，如果它有 29 美元，那么它会很高兴在一个公平的方式下，如抛掷一枚硬币，下注 19 美元去赢取另一个 Agent 的 1 美元。该 Agent 不想要超过 60 美元的玩具，因为这会使它认为这是在敲诈。

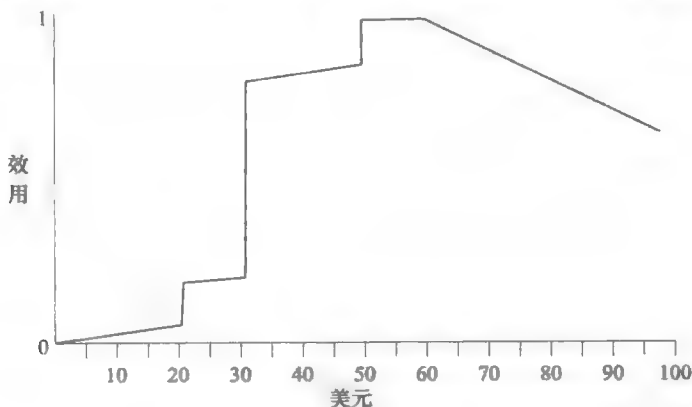


图 9-3 来自例 9-2 的一个金钱-效用均衡曲线

因子化效用

正如前面的定义,效用是结果或状态的一个函数。通常,由于存在的状态过多,以至于难以直接用状态来表示效用函数。使用特征来表示效用函数是一个更容易的方式。

假设每一个结果都能使用特征 X_1, \dots, X_n 来描述。**累加效用**(additive utility)能够分解为因子的集合:

$$u(X_1, \dots, X_n) = f_1(X_1) + \dots + f_n(X_n)$$

378 这样的分解构成了**累加独立性**(additive independence)假设。

既然效用可以累加,那就极大地简化了**偏好引出**(preference elicitation)——获取用户的偏好的问题。注意,效用的分解不是唯一的,因为添加一个常数到一个因子上,同时从另一个因子上减去相应的效用,可以得到同样的效用。为了将其分解为规范的形式,我们可以定义一个局部效用函数 $u_i(X_i)$, 当 X_i 的取值是其最差值时, $u_i(X_i)$ 的取值为 0; 当 X_i 的取值是其最优值时, $u_i(X_i)$ 的取值为 1; 同时还有一系列取值为非负数的权重 w_i , 所有的权重之和为 1, 且满足:

$$u(X_1, \dots, X_n) = w_1 \times u_1(X_1) + \dots + w_n \times u_n(X_n)$$

为引出这样的一个效用函数,需要引出每一个局部效用函数并估算其权值。对于每一个特征,只要它是相关的,就必然有一个最优的取值和最差的取值。可以使用如下的方法估算局部函数及其权值。我们仅考虑 X_1 , 对其他特征的考察与此类似。对于特征 X_1 , 其取值为 x_1 和 x'_1 , 并且 X_2, \dots, X_n 分别取值为 x_2, \dots, x_n :

$$u(x_1, x_2, \dots, x_n) - u(x'_1, x_2, \dots, x_n) = w_1 \times (u_1(x_1) - u_1(x'_1)) \quad (9.1)$$

当 x_1 是最好的结果而 x'_1 是最坏的结果时(此时 $u_1(x_1) - u_1(x'_1) = 1$), 权重 w_1 能够直接计算出来。对于域 X_1 的其他取值,其对应的效用 u_1 可通过式(9.1)计算——此时 x'_1 是 X_1 的最差取值($u_1(x'_1) = 0$)。

对期望效用的质疑

对于期望效用理论,存在着多种质疑。Allais 和 Hagen 在 1953 年提出的 Allais 悖论 [Allais, Hagen, 1979] 如下。下面两个选项中,你更愿意选哪个?

A: \$1m——一百万美元

B: 抽奖法[0.10: \$2.5m, 0.89: \$1m, 0.01: \$0]

类似地,你会在下面两个选项中选哪个?

C: 抽奖法[0.11: \$1m, 0.89: \$0]

D: 抽奖法[0.10: \$2.5m, 0.9: \$0]

结果是许多人选 A 而不选 B, 选 D 而不选 C。这个选择结果与前面的理性公理不一致。为解释这个现象,上述选项可以合并为同样的形式:

A, C: 抽奖法[0.11: \$1m, 0.89: X]

B, D: 抽奖法[0.10: \$2.5m, 0.01: \$0, 0.89: X]

在选项 A 和 B 中, X 是一百万美元。在选项 C 和 D 中, X 是 0 美元。仅合并各选项的不同部分看起来是一个合适的策略,但人们似乎倾向于确定的结果。

Tversky 和 Kahneman[1974]通过一系列的人群实验,说明了人们是如何系统化地偏离效用理论的。造成这种偏差的一个原因是问题描述的**框架效应**(framing effect)。考虑下面的问题:

- 一种疾病预计会致 600 人死亡。有下面两种应对方案：

方案 A：挽救 200 人

方案 B：有 1/3 的概率挽救 600 人，或者有 2/3 的概率导致 600 人全部死亡

你愿意选择哪个方案？

- 一种疾病预计会致 600 人死亡。有下面两种应对方案：

方案 C：有 400 人死亡

方案 D：以 1/3 的概率致 0 人死亡，有 2/3 的概率致 600 人死亡

你愿意选哪个方案？

Tversky 和 Kahneman 的研究表明，在 A 和 B 方案中，有 72% 的人选择 A；在 C 和 D 方案中，有 22% 的人选择 C。然而，这两个方案是完全一样的，只不过描述方式不同罢了。

处理预期效用的另一个方法是 Kahneman 和 Tversky 提出的**前景理论**(prospect theory)，在这个理论中考虑了 Agent 当前的财富状况。也就是说，做决策的依据是 Agent 的收益和损失，而不是结果。然而，这样做的原因仅是它更符合人类的选择习惯，而不意味着这对于人工 Agent 来说是最好的选择。但是，如果一个人工 Agent 必须与人类交互，则它应该考虑人类的推理方式。

380

累加独立性假设是一个强独立假设。特别的，在式(9.1)中，对于所有的 X_2, \dots, X_n 的取值 x_2, \dots, x_n ，计算效用的差异的方法是相同的。

累加独立性假设常常不是一个好的假设。对于两个二值特征的两个取值，如果同时取两个值的效用比分别取两个值的效用之和低，则称这两个取值是**互补**(complement)的。假设有特征 X 和 Y ，其值域分别为 $\{x_0, x_1\}$ 和 $\{y_0, y_1\}$ 。对于 x_1 和 y_1 ，如果 Agent 在已经有其中一个取值的情况下又有了另一个取值时的效用优于 Agent 有其中一个取值而没有另一个时的效用，则称 x_1 和 y_1 是互补的：

$$u(x_1, y_0) - u(x_0, y_0) < u(x_1, y_1) - u(x_0, y_1)$$

另外，这也蕴含着 y_1 和 x_1 是互补的。

对于两个二值特征的两个取值，如果同时取两个值的效用比分别取两个值的效用之和低，则称这两个取值是**替换**(substitute)的。假设 x_1 和 y_1 是替换的，则意味着 Agent 在已经有其中一个取值的情况下又有了另一个取值时的效用要低于 Agent 有其中一个取值而没有另一个时的效用：

$$u(x_1, y_0) - u(x_0, y_0) > u(x_1, y_1) - u(x_0, y_1)$$

这也蕴含着 y_1 和 x_1 也是替换的。

【例 9-3】 对于一个旅游领域的采购 Agent 来说，预定某一特定日期的机票，和预定同一天的酒店是互补的：两者只取其一不会得到好的结果。

考虑到某人很乐意在度假的一天中外出旅游一次(不是两次)，则安排在同一天的两个不同的旅游是可以替换的。不过，如果两个旅游地相距很近，并且去两地的旅行时间较长，则两次旅游有可能是互补的(如果旅游两地，则更体现了旅行时间的价值)。

累加效用假设不存在效用的替换和互补。当有交互时，我们就需要一个更复杂的模型，比如**广义累加独立性模型**(generalized additive independence)，在其中将效用表示为各因子之和。这与 4.10 节中的优化模型相似。无论如何，我们希望使用这些模型来计算期望的效用。引出广义累加独立性模型要比引出累加模型更为有用，因为一个特征能够出现在许多因子上。

9.2 一次性的决策

应用于 Agent 的基本决策理论依赖于下面的假设：

- Agent 知道它能做出哪些动作。
- 执行每一个动作所产生的效果可以用结果上的概率分布来描述。
- 一个 Agent 的偏好由结果的效用来表示。

如果 Agent 仅执行一步，那么一个理性的 Agent 应该选择使得期望效用最大的那个动作——这是命题 9.3 的一个结果。

【例 9-4】 考虑传送机器人的一个问题：机器人的行动产生的结果是不确定的。特别地，考虑从图 3-1 中的位置 o109 到达 mail 位置的问题，此时机器人有可能偏离原定路线并从楼梯跌下去。假设机器人能够使用垫子（垫子虽不能减少事故的发生概率，但能够减轻事故的严重性）。不幸的是，这些垫子增加了机器人的重量。机器人也可以选择绕一个大弯子，这样可以减少事故发生的概率，但走完全程需要更多的时间。

因此，机器人不得不决定是否要使用垫子，还要决定走哪条路（较远的路还是较近的路）。是否会发生事故是不在其控制之下的事情，尽管它可以通过选择走较远的路程来减少事故发生的概率。对于 Agent 的每组选择和是否发生事故的情况，其结果都介于发生严重损坏和在不增加重量的情况下快速到达之间。

为了给一次性的决策建模，可以使用一个**决策变量**（decision variable）来表示 Agent 的选择。一个决策变量就像一个随机变量一样，有一个值域，但没有一个与之关联的概率分布。相反，由 Agent 为决策变量选择一个值。一个**可能情景**（possible world）限定了随机变量和决策变量的值域，而对于每一个决策变量的值域，其对应的随机变量都有一个概率分布。也就是说，对于每一个决策变量的每一次赋值，满足该次赋值的可能情景的度量之和为 1。条件概率只有在决策变量的取值是其依赖条件的一部分时才有定义。

图 9-4 所示的**决策树**描述了 Agent 可选的各种选项及其所产生的结果（这里与用于分类的决策树不同）。阅读这个决策树时需要从根节点（在图的最左边）开始。对于每一个节点，可从其后续的分支中选择一个来继续下一个选择。对于决策节点（用矩形表示的节点），Agent 可以做出选择，确定按照哪个分支走下去。对于每一个随机节点（用圆圈表示的节点），Agent 不能选择沿哪个分支走下去，此时，在可选分支中均分配有一定的选择概率。通往一个叶子的每一条路径对应于一个情景 w_i ，它表示相应路径所能产生的结果。

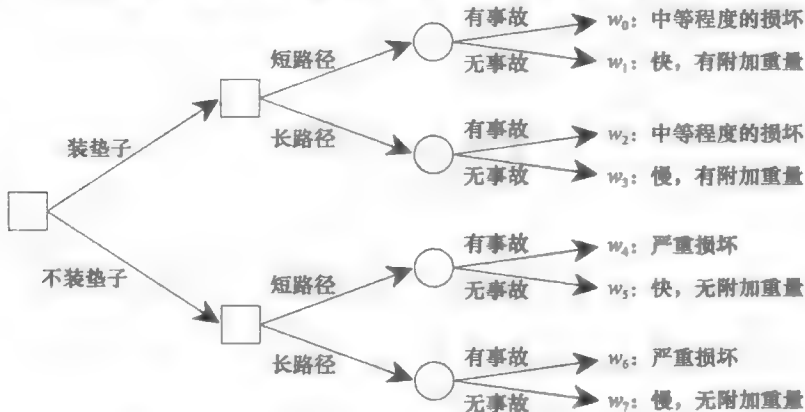


图 9-4 传送机器人的一个决策树。矩形节点表示机器人能够做出的决策。圆圈节点表示机器人在做决策之前无法观察到的随机因素

【例 9-5】 在例 9-4 中共有两个决策变量，一个对应于决定机器人是否使用垫子，另一个对应于决定走哪条路径。有一个随机变量，它确定是否发生事故。共存在 8 个可能情景，对应于图 9-4 决策树中的 8 条路径。

Agent 应该做出什么决定取决于快速到达这一要求的紧迫程度、垫子的重量、为减少发生严重事故至中等事故造成的损失而付出的代价及发生事故的可能性大小。

382

命题 9.3 的证明明确了对于结果的愿望程度的度量方法。假设我们决定效用的取值范围是 $[0, 100]$ 。首先，选择最好的结果，即 w_5 ，并指定其效用为 100；最差的结果是 w_6 ，因此指定其效用为 0。对于其他的每一个情景，可考虑为介于 w_6 与 w_5 之间的一次抽奖。例如， w_0 可能具有的效用为 35，意味着 Agent 对于 w_0 和 $[0.35; w_5, 0.65; w_6]$ 是同等对待、没有偏好的。结果 w_0 比结果 w_2 稍好，而 w_2 的效用可设为 30。结果 w_1 的效用可设为 95，因为它只比结果 w_5 稍差。

【例 9-6】 在疾病诊断中，决策变量对应于各种治疗和检查。效用的大小依赖于检查和治疗的成本，病人的情况是好转、不变还是死亡，病人有短期痛苦还是有长期痛苦。治疗病人的结果依赖于病人接受的治疗、病人的生理及疾病的具体情况（可能有些具体情况无法知晓）。尽管我们这里使用了医疗诊断的术语，但这个方法对于人工制品（比如飞机）的诊断同样适用。

在一个一次性的决策中，Agent 为每一个决策变量选择一个值。这可以通过视所有决策变量为一个组合决策变量的方法来建立模型。该组合决策变量的值域为每一个决策变量的值域的叉乘。该组合决策变量记为 D 。

每一个情景 ω 对应决策变量 D 的一个赋值，同时对应每一个随机变量的赋值。

383

一个**单一决策** (single decision) 是给决策变量的一次赋值。单一决策 $D = d_i$ 的**期望效用** (expected utility) 是：

$$e(U | D = d_i) = \sum_{\omega \models (D=d_i)} U(\omega) \times P(\omega)$$

其中 $P(\omega)$ 是情景 ω 的发生概率， $U(\omega)$ 是情景 ω 的效用 U 的取值； $\omega \models (D = d_i)$ 的含义是决策变量 D 在情景 ω 下取值 d_i 。因此，期望效用的计算就是已选择合适决策的情景之和。

最优单一决策 (optimal single decision) 是期望效用最大的决策，即 $D = d_{\max}$ 是一个最优决策，如果有：

$$e(U | D = d_{\max}) = \max_{d_i \in \text{dom}(D)} e(U | D = d_i)$$

其中 $\text{dom}(D)$ 是决策变量 D 的值域。因此有：

$$d_{\max} = \arg \max_{d_i \in \text{dom}(D)} e(U | D = d_i)$$

【例 9-7】 例 9-4 中的传送机器人问题是一个单一决策问题。在该问题中，机器人不得不决定是否使用垫子的变量 Wear_Pads 和指示走哪条路的变量 Which_Way 的取值。单一决策是决策变量的复合 $\langle \text{Wear_Pads}, \text{Which_Way} \rangle$ 。对每一个决策变量的每一次赋值都有一个期望的值。例如， $\text{Wear_Pads} = \text{true} \wedge \text{Which_Way} = \text{short}$ 的期望效用是：

$$\begin{aligned} e(U | \text{wear_pads} \wedge \text{Which_Way} = \text{short}) \\ = P(\text{accident} | \text{wear_pads} \wedge \text{Which_way} = \text{short}) \times \text{utility}(w_0) \\ + (1 - P(\text{accident} | \text{wear_pads} \wedge \text{Which_way} = \text{short})) \times \text{utility}(w_1) \end{aligned}$$

其中情景 w_0 和 w_1 在图 9-4 中有解释， wear_pads 的含义是 $\text{Wear_Pads} = \text{true}$ 。

单阶段决策网络

决策树是一个基于状态的表示，因为各个情景对应着最终的状态。然而，使用特征(用变量表示)来表示和推理决策过程更加自然高效。

单阶段决策网络(single-stage decision network)是信念网络的一个扩展，它有三类节点：

- **决策节点**，用矩形表示，代表了决策变量。Agent 为每一个决策变量选择一个取值。对于有多个决策变量的情况，我们假设决策节点有一个完整的顺序，并且称该顺序中出现在决策节点 D 之前的决策节点为节点 D 的父节点。
- **机会节点**，用椭圆表示，代表了随机变量。这些节点的表示与信念网络中节点的表示方法一致。每一个机会节点有一个相关联的值域及一个由其父节点给出的条件概率。与信念网络一样，机会节点的父节点表示条件依赖：给定了父节点后，一个变量独立于其子孙节点。在一个决策网络中，机会节点和决策节点都可以作为机会节点的父节点。
- **效用节点**(utility node)，用菱形表示，代表了效用。效用节点的父节点是效用所依赖的变量。机会节点和决策节点都可成为效用节点的父节点。

每一个机会变量和每一个决策变量都有一个关联的值域，但没有值域与效用节点相关联。虽然机会节点代表了随机变量，决策节点代表了决策变量，但却没有效用变量与效用节点相联系。效用节点为其父节点提供了一个函数。

与一个决策网络相联系的有为每一个机会节点设置的条件概率(由其父节点提供，与信念网络中的情况一样)，还有作为效用节点的父节点的函数，该函数用于表示效用。在决策网的说明中，没有表格与决策节点相联系。

【例 9-8】 图 9-5 给出了例 9-4 的一个决策网络的表示。这里共需做出两个决策：走哪条路和是否戴垫子。是否发生事故仅取决于走哪条路。效用依赖于上述三个变量。

这个决策网络需要两个因子：一个代表了发生事故的条件概率 $P(\text{Accident} | \text{WhichWay})$ ，另一个代表了效用(该效用是变量 WhichWay 、 Accident 和 WearPads 的函数)。图 9-5 中给出了这两个因子的取值。

单阶段决策网络的一个**策略**(policy)是对每一个决策变量的一种赋值。每一个策略都有一个**期望效用**，即在该策略的条件下，效用的条件期望值。一个**最优策略**(optimal policy)是期望效用最大的策略，也就是说，不存在其他的、具有更高期望效用的策略。

图 9-6 说明了在单阶段决策网络中如何使用**变量消除**的方法来发现一个最优策略。在剪去不相关的节点和综合所有随机变量之后，将只剩下一个代表了每一个决策变量组合的期望效用的因子。这个因子不必是在所有决策变

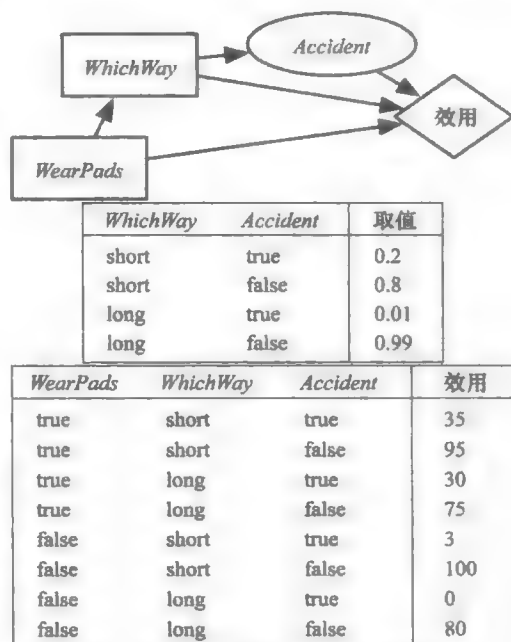


图 9-5 传送机器人的单阶段决策网络

量之上的一个因子, 不过, 那些不包含在内的决策变量与最终决策也不会有关系。

```

1: procedure OptimizeSSDN(DN)
2:   Inputs
3:     DN, 一个单阶段决策网络
4:   Output
5:     一个最优策略及其期望效用
6:   剪去所有不是效用节点的祖先节点的节点;
7:   概括消除所有的机会节点;
8:   ——在这个阶段, 只剩下一个由效用导出的因子  $F$ ;
9:   记  $F$  中的最大取值为  $v$ ;
10:  记使效用的取值最大的赋值方案为  $d$ ;
11:  返回  $d$  和  $v$ 

```

图 9-6 单阶段决策网络的变量消除

【例 9-9】考虑在图 9-5 所示的决策网络上运行 *OptimizeSSDN*。没有节点可被剪去, 因此只有随机变量 *Accident* 可以合并求和。为了这样做, 该网络上的两个因子需要乘起来 (因为它们都包含有变量 *Accident*), 并消去 *Accident*, 最后得到下面的因子:

<i>WearPads</i>	<i>WhichWay</i>	取值
true	short	$0.2 * 35 + 0.8 * 95 = 83$
true	long	$0.01 * 30 + 0.99 * 75 = 74.55$
false	short	$0.2 * 3 + 0.8 * 100 = 80.6$
false	long	$0.01 * 0 + 0.99 * 80 = 79.2$

因此, 具有最大值的策略, 即最优策略, 是走较短路径并戴垫子, 其期望效用是 83。◀

9.3 序贯决策

一般来说, 在对其所处环境一无所知的情况下, Agent 既不会作出多个决策, 也不会作出哪怕一个决策。Agent 作出决策的典型情景是这样的: Agent 根据其所处环境的观察决定要做出一个动作, 执行那个动作, 观察动作结果对环境的影响, 然后根据观察作下一个决策, 依此类推。后续的动作基于其所观察到的内容及前一动作对环境所产生的影响。在这个情境下, 执行一个动作的唯一原因是为将来的动作提供信息。

序贯决策问题(sequential decision problem)是一系列的决策, 其中需要为每一个决策考虑如下问题:

- Agent 能够做哪些动作。
- Agent 不得不开始执行动作时, 它能够获得或将会获得的信息。
- 每个动作对环境产生的影响。
- Agent 对上述影响的期望程度。

【例 9-10】考虑一个简单的诊断情况: 医生首先选择为病人做一些检查, 然后根据这些检查的结果制定病人的治疗方案。医生考虑决定做某项检查的原因是这个检查的结果能够为治疗的下一阶段提供支持。检查的结果可能作为制定治疗方案时的信息, 而不能作为决定做该检查时的信息。做检查通常是一个好的做法, 即使该检查会伤害病人。

医生可选择的行为是可供检查的项目和可供选择的治疗方案。对于做出检查的决策来

说, 可用的信息将是病人表现出来的症状。对于制定治疗方案来说, 可用的信息包括病人的症状、已检查的项目及这些检查的结果。检查的影响是检查的结果, 其取决于检查的项目和病人的疾病。治疗的影响是治疗所能起到的一些作用及病人的疾病。效用包括检查的成本、短期带给病人的痛苦和不便及长期的预后情况。

9.3.1 决策网络

决策网络(decision network, 也称影响图(influence diagram))是有限的序贯决策问题的图形化表示。决策网络扩展了信念网络以包含决策变量和效用, 也扩展了单阶段决策网络以允许序贯决策。

387

特别的, 决策网络是一个有向无环图(DAG), 其中有机会节点、决策节点和效用节点。它对单阶段决策网络进行了扩展, 使得机会节点和决策节点都可成为决策节点的父节点。决策节点的人边表示在作决策时那些可用的信息。机会节点的人边表示概率依赖关系。效用节点的人边表示效用所依赖的内容。

无遗忘(no-forgetting)Agent 的决策是完全有序的, 并且记得它之前做出的决策及做出那些决策时可用的信息。无遗忘 Agent 网络中的决策节点是完全有序的; 如果决策节点 D_i 在决策节点 D_j 之前, 则有 D_i 是 D_j 节点的父节点, 且 D_i 的所有父节点也是 D_j 的父节点。因此, 在 D_i 节点已知的所有信息在 D_j 节点上同样有效, 且在 D_i 节点选择的动作也成为 D_j 节点可用信息的一部分。无遗忘条件足以确保下面的定义是有意义的, 且下面的算法能够工作。

【例 9-11】 图 9-7 是 Agent 外出时用以决定是否带雨伞的一个简单的决策网络。Agent 的效用取决于天气情况和它是否带了雨伞。然而, Agent 并不观察天气情况, 而是仅观察天气预报的情况。而天气预报的情况条件依赖于天气情况。

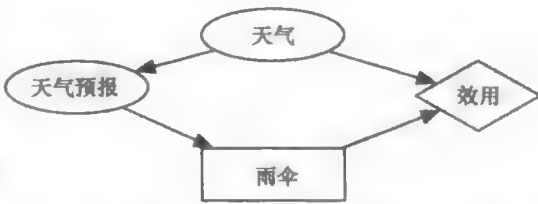


图 9-7 根据天气情况决定是否带雨伞的决策网络

作为决策网络的一部分, 设计者必须明确每一个随机变量的值域及每一个决策变量的值域。假设随机变量 *Weather* 的值域是 {no-rain, rain}, 随机变量 *Forecast* 的值域是 {sunny, rainy, cloudy}, 决策变量 *Umbrella* 的值域是 {takeIt, leaveIt}。效用节点没有值域。设计者还必须明确随机变量在其父节点的作用下的发生概率。假设 $P(\textit{Weather})$ 的定义如下:

$$P(\textit{Weather} = \textit{rain}) = 0.3$$

而 $P(\textit{Forecast} | \textit{Weather})$ 的定义如下:

<i>Weather</i>	<i>Forecast</i>	<i>Probability</i>
<i>norain</i>	<i>Sunny</i>	0.7
<i>norain</i>	<i>cloudy</i>	0.2
<i>norain</i>	<i>Rainy</i>	0.1
<i>rain</i>	<i>Sunny</i>	0.15
<i>rain</i>	<i>cloudy</i>	0.25
<i>rain</i>	<i>Rainy</i>	0.6

388

假设效用函数 $Utility(Weather, Umbrella)$ 如下:

Weather	Umbrella	Utility
norain	takeIt	20
norain	leaveIt	100
rain	takeIt	70
rain	leaveIt	0

对于决策变量 $Umbrella$, 没有相应的取值列表。确定该变量的取值是规划者的任务, 且依赖于天气预报的情况。

【例 9-12】图 9-8 是表示例 9-10 的场景的一个决策网络。病人症状取决于疾病。要做哪些检查取决于有哪些症状。检查结果取决于疾病和所做的检查。治疗取决于症状、做了哪些检查及检查结果。结果依赖于疾病和治疗。效用依赖于检查的成本和副作用, 以及治疗的结果。

注意, 做出检查决策和治疗决策的诊断处理并不能确切地知道病人得了什么病, 除非检查结果能够非常确定这一点(然而, 通常都不是这种情况)。

【例 9-13】图 9-9 是扩展了图 6-1 信念网络的决策网络。Agent 能够接收到人们离开建筑物的报告, 并必须做出是否打电话给消防部门的决定。在打电话之前, Agent 能够检查烟雾情况, 但这需要付出一些成本。效用依赖于是否打了电话、是否有火情及检查烟雾情况的成本。

在这个序贯决策问题中, 需要作出两个决策。首先, Agent 必须决定是否检查烟雾。是否有人离开建筑物的报告是作这项决策前可以获得的信息。其次, Agent 必须做出是否给消防部门打电话的决策。在作这个决策时, Agent 将需知道是否有一个报告、是否检查了烟雾及是否看到了烟雾。假设所有这些变量都是二值的。

决策网络必须的信息包括信念网络的条件概率, 以及

- $P(SeeSmoke|Smoke, CheckSmoke)$: 如何看到烟雾取决于 Agent 是否检查有烟雾及是否有烟雾。假设 Agent 有一个工作完好的烟雾传感器。此时, 当且仅当检查了是否有烟雾并且确实有烟雾时, Agent 就能看到烟雾。(参看习题 9.6)
- $Utility(CheckSmoke, Fire, Call)$: 效用的情况取决于 Agent 是否检查了烟雾、是否有火情及是否给消防部门打了电话。图 9-10 列出了效用的信息。这个效用函数表明打电话的成本是 200、检查是否有烟雾的成本是 20、不过在有火情而没有打电话的情况下成本是 5000。效用是成本的负数。

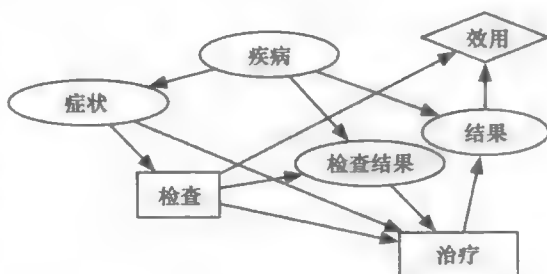


图 9-8 医生诊断的决策网络

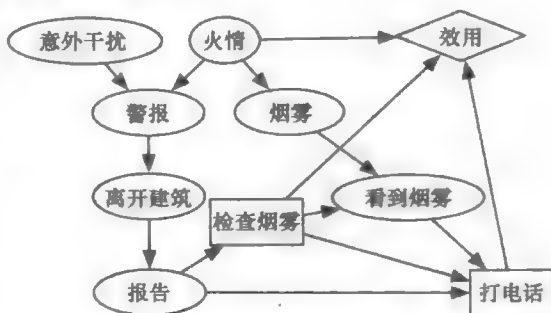


图 9-9 报警问题的决策网络

CheckSmoke	Fire	Call	Utility
yes	yes	call	-220
yes	yes	do not call	-5 020
yes	no	call	-220
yes	no	do not call	-20
no	yes	call	-200
no	yes	do not call	-5 000
no	no	call	-200
no	no	do not call	0

图 9-10 报警决策网络的效用

9.3.2 策略

策略具体说明 Agent 在所有不可预见的情况下该做什么。Agent 希望找到一个最优的策略，即能够最大化期望效用的策略。

一个策略由对应于每一个决策变量的决策函数组成。一个决策变量的决策函数(decision function)是对于决策变量的父节点变量的每一个赋值而明确决策变量取值的一个函数。因此，一个策略就明确规定了 Agent 在面对所有可能的取值时应该采取的行动。

【例 9-14】 在例 9-11 中，有一些策略如下：

- 总是携带雨伞。
- 只在天气预报为 *rainy* 时才带雨伞。
- 只在天气预报为 *sunny* 时才带雨伞。

共有 8 种不同的策略，原因是天气预报只有三种情况，且对于每一种情况只有两个选择。

【例 9-15】 在例 9-13 中，一个策略需要明确 *CheckSmoke* 的决策函数和 *Call* 的决策函数。其中一些策略如下：

- 从不检查烟雾情况，在接到报告后打电话。
- 总是检查烟雾情况，在看到烟雾后打电话。
- 接到报告后才检查烟雾情况，在接到报告并看到烟雾后打电话。
- 没有接到报告则检查烟雾情况，没有看到烟雾则打电话。
- 总是检查烟雾情况，从不打电话。

在这个例子中，共有 1024 个不同的策略(因为每个变量都是二值的)。对于检查烟雾变量 *CheckSmoke*，共有 4 个决策函数。对于打电话变量 *Call* 共有 2^6 个决策函数；对于 *Call* 变量的 8 个父节点变量的任意取值，Agent 可选择打电话也可选择不打电话。

策略的期望效用

可以通过确定 Agent 执行策略的期望效用来评价策略的优劣。一个理性的 Agent 应采纳最大化其期望效用的策略。

一个可能情景(possible world)为每一个随机变量和决策变量确定其取值。一个可能情景不具有发生概率，直到确定了所有的决策变量的取值。如果一个可能情景中的每一个决策变量的取值等于一个策略对应的决策函数为每个决策变量选定的取值，则称该可能情景满足该策略。如果 ω 是一个可能情景， π 是一个策略，则 $\omega \models \pi$ 定义为可能情景 ω 满足策略 π 。

一个可能情景对应着一个完整的动作顺序的历史选择记录及所有随机变量和决策变量(包括所有观察到的变量)的确切取值，认识到这一点很重要。在 Agent 采用策略 π 时，如果可能情景 ω 是一个可能历史的演变，则它满足策略 π 。可满足性约束强化了这样的直觉：Agent 实际上将执行策略 π 为每一个可能的观察规定的动作。

策略的期望效用为：

$$\epsilon(\pi) = \sum_{\omega \models \pi} U(\omega) \times P(\omega)$$

其中 $P(\omega)$ 表示情景 ω 发生概率，其取值为在各机会节点的父节点给出的条件概率的前提下各机会节点的概率的乘积； $U(\omega)$ 表示情景 ω 下的效用。

【例 9-16】在例 9-11 中，设 π_1 为如下策略：如果天气预报是多云则带雨伞，否则不带伞。该策略的期望效用是满足该策略的所有情景的效用的平均：

$$\begin{aligned} \epsilon(\pi_1) = & P(\text{norain})P(\text{sunny}|\text{norain})\text{Utility}(\text{norain}, \text{leaveIt}) \\ & + P(\text{norain})P(\text{cloudy}|\text{norain})\text{Utility}(\text{norain}, \text{takeIt}) \\ & + P(\text{norain})P(\text{rainy}|\text{norain})\text{Utility}(\text{norain}, \text{leaveIt}) \\ & + P(\text{rain})P(\text{sunny}|\text{rain})\text{Utility}(\text{rain}, \text{leaveIt}) \\ & + P(\text{rain})P(\text{cloudy}|\text{rain})\text{Utility}(\text{rain}, \text{takeIt}) \\ & + P(\text{rain})P(\text{rainy}|\text{rain})\text{Utility}(\text{rain}, \text{leaveIt}), \end{aligned}$$

其中 norain 表示 $\text{Weather} = \text{norain}$ ， sunny 表示 $\text{Forecast} = \text{sunny}$ ，以此类推。注意，决策变量的取值由策略选定，且仅依赖于天气预报。

最优策略 π^* 是对于所有的策略 π 有 $\epsilon(\pi^*) \geq \epsilon(\pi)$ ，即一个最优策略是其期望效用最大的策略。

假设一个二值的决策节点有 n 个二值的父节点，则对父节点的不同赋值共有 2^n 种情况，对决策节点来说共有 2^{2^n} 个不同的决策函数。策略的数量等于各个决策变量的不同的决策函数的数量之积。这样，即便是很小的例子也会有巨量的策略。因此，通过枚举策略集合来查找最优策略的算法非常低效。

9.3.3 决策网络的变量消除

幸运的是，我们不必枚举全部的策略，而是可以使用变量消除(variable elimination, VE)的方法来找到最优策略。该方法的思想是首先考虑最后一个决策，查找其父节点的每一个取值的最优决策，并生成产生这些最大取值的一个因子。这样就形成了一个新的决策网络，但少了一个决策。如此迭代执行，最终求解问题。

图 9-11 给出了在决策网络中使用 VE 的方法。实际上它计算了一个最优决策的期望效用。该算法根据某消除排序结果消除了不作为决策节点的父节点的随机变量。消除排序不影响结果的正确性，使用排序的目的是提高效率。

392

```

1: procedure VE_DN(DN):
2:   Inputs
3:   DN, 一个单阶段决策网络
4:   Output
5:   一个最优策略及其期望效用
6:   Local
7:   DFs: 决策函数的集合, 初始为空
8:   Fs: 因子的集合
9:   删去效用节点的所有非父节点的节点
10:  在集合 Fs 中为每一个条件概率创建一个因子
11:  在 Fs 中为效用创建一个因子
12:  while 仍然剩有决策节点 do
13:    合并不作为决策节点的父节点的节点的随机变量
14:    ▷ 在这个阶段, 因子 F 中的一个决策节点 D 附带有其父节点的一个子集
15:    把 max_D F 加入 Fs 中
16:    将 arg max_D F 加入 DFs 中
17:  合并所有剩余的随机变量
18:  返回 DFs 和剩余因子的乘积

```

图 9-11 决策网络的变量消除

在消除了所有没有作为决策节点的父节点的随机变量之后，必然存在一个包含在一个因子中的决策变量及其父节点的某个子集，原因是 Agent 是无遗忘的。这是行为顺序中的最后一个动作。

为消除决策节点，VE_DN 算法中为此决策选择使效用最大的那个值。这个最大化在剩余的变量上生成了一个新的因子，及为消除掉的决策变量生成的一个决策函数。这个用最大化的方法生成的决策函数是一个最优策略的组成部分。

393

【例 9-17】 在例 9-11 中，共有三个初始的因子，分别表示天气的概率 $P(\text{Weather})$ 、天气预报的条件概率 $P(\text{Forecast} | \text{Weather})$ 和效用函数 $\text{Utility}(\text{Weather}, \text{Umbrella})$ 。首先消去 Weather，方法是将三个因子相乘，然后合并 Weather 并给 Forecast 和 Umbrella 生成一个因子：

Forecast	Umbrella	Value
sunny	takeIt	12.95
sunny	leaveIt	49.0
cloudy	takeIt	8.05
cloudy	leaveIt	14.0
rainy	takeIt	14.0
rainy	leaveIt	7.0

为通过 Umbrella 来最大化因子，针对每一个 Forecast 的取值，VE_DN 算法选择使得因子最大的 Umbrella 的取值。例如，当 Forecast 的取值是 sunny 时，Agent 应该把雨伞放在家里，因为此时因子的值是 49.0。

VE_DN 算法为 Umbrella 建立一个最优的决策函数，该函数为 Umbrella 选择的值对于每一个 Forecast 的取值都会使因子的取值最大：

Forecast	Umbrella
sunny	leaveIt
cloudy	leaveIt
rainy	takeIt

此时会生成一个新的因子，该因子的取值是上面对应于每一个 Forecast 取值的因子：

Forecast	Value
sunny	49.0
cloudy	14.0
rainy	14.0

现在根据上述因子合并计算 Forecast，得到的取值是 77.0。这就是最优策略的期望效用。◀

【例 9-18】 考虑例 9-13。在合并变量之前有如下的因子：

Meaning	Factor
$P(\text{Tampering})$	$f_0(\text{Tampering})$
$P(\text{Fire})$	$f_1(\text{Fire})$
$P(\text{Alarm} \text{Tampering}, \text{Fire})$	$f_2(\text{Tampering}, \text{Fire}, \text{Alarm})$
$P(\text{Smoke} \text{Fire})$	$f_3(\text{Fire}, \text{Smoke})$
$P(\text{Leaving} \text{Alarm})$	$f_4(\text{Alarm}, \text{Leaving})$
$P(\text{Report} \text{Leaving})$	$f_5(\text{Leaving}, \text{Report})$
$P(\text{SeeSmoke} \text{CheckSmoke}, \text{Smoke})$	$f_6(\text{Smoke}, \text{SeeSmoke}, \text{CheckSmoke})$
$\text{utility}(\text{Fire}, \text{CheckSmoke}, \text{Call})$	$f_7(\text{Fire}, \text{CheckSmoke}, \text{Call})$

只要选择了合适动作，则期望效用就是发生概率与相应效用的乘积。

VE_DN 算法合并不作为决策节点的父节点的随机变量。因此，需合并 *Tampering*、*Fire*、*Alarm*、*Smoke* 和 *Leaving* 变量。在消除这些变量之后，只剩下一个因子，该因子的部分内容如下(保留两位小数)：

<i>Report</i>	<i>SeeSmoke</i>	<i>CheckSmoke</i>	<i>Call</i>	<i>Value</i>
t	t	t	t	-1.33
t	t	t	f	-29.30
t	t	f	t	0
t	t	f	f	0
t	f	t	t	-4.86
t	f	t	f	-3.68
...

根据这个因子，可通过为 *Call* 选择一个取值使得对变量 *Report*、*SeeSmoke* 和 *CheckSmoke* 的每一组赋值都使 *Value* 的取值最大的方法为 *Call* 创建一个最优决策函数。数值 -1.33 和 -29.3 的较大者为 -1.33，因此，当 *Report* = t, *SeeSmoke* = t 且 *CheckSmoke* = t 时，最优的动作是 *Call* = t (此时的因子为 -1.33)。对于 *Report*、*SeeSmoke* 和 *CheckSmoke* 的其他取值，使用同样的方法。

Call 的一个最优决策函数是：

<i>Report</i>	<i>SeeSmoke</i>	<i>CheckSmoke</i>	<i>Call</i>
t	t	t	t
t	t	f	t
t	f	t	f
...

请注意，当 *SeeSmoke* = t、*CheckSmoke* = f 时，*Call* 取任何值都可以。在这种情况下，Agent 计划做什么都无关紧要，因为这种情况实际上不会出现。

由最大化 *Call* 而产生的因子包括了针对 *Report*、*SeeSmoke* 和 *CheckSmoke* 的各种取值的最大因子取值：

<i>Report</i>	<i>SeeSmoke</i>	<i>CheckSmoke</i>	<i>Value</i>
t	t	t	-1.33
t	t	f	0
t	f	t	-3.68
...

然后可以合并 *SeeSmoke*，得到如下的因子：

<i>Report</i>	<i>CheckSmoke</i>	<i>Value</i>
t	t	-5.01
t	f	-5.65
f	t	-23.77
f	f	-17.58

395

针对 *Report* 最大化 *CheckSmoke* 得到决策函数和因子如下：

<i>Report</i>	<i>CheckSmoke</i>
t	t
f	f

<i>Report</i>	<i>CheckSmoke</i>
t	-5.01
f	-17.58

合并 *Report* 得到期望效用 -22.60(考虑了舍入误差)。

因此，返回的最优策略可列如下：

```
checkSmoke ← report
call_fire_department ← see_smoke
call_fire_department ← report ∧ ¬check_smoke ∧ ¬see_smoke
```

最后一个规则永远也不会用到，因为 Agent 根据最优策略会在接收到报告时检查烟雾情况。不过，在执行 *VE_DN* 算法时，Agent 在优化 *Call* 时不知道关于 *CheckSmoke* 的最优策略。Agent 决定是否要检查烟雾时，仅考虑可用烟雾信息做什么。

还需要注意，在这种情况下，尽管检查烟雾情况需要付出一些成本，但是这个检查是值得的，因为检查获得的信息很有价值。

下面这个例子说明了，在 *VE* 算法优化决策时，所得的因子中的决策变量是如何包含其父节点的一个子集的。

【例 9-19】 考虑例 9-11，不过增加了一个从 *Weather* 到 *Umbrella* 的边，也就是说，Agent 不仅要观察天气情况，而且也要考察天气预报的情况。此时，没有需要合并的随机变量，从而生成的包含了决策节点和其父节点的一个子集的因子就是原始的效用因子。因此，使 *Umbrella* 最大化的决策函数和因子如下：

<i>Weather</i>	<i>Umbrella</i>	<i>Weather</i>	<i>Value</i>
<i>norain</i>	<i>leaveIt</i>	<i>norain</i>	100
<i>rain</i>	<i>takeIt</i>	<i>rain</i>	70

请注意，天气预报的情况与决策毫无关系。知道天气预报的情况不能为 Agent 提供任何有用的信息。

合并 *Forecast* 得到的因子均为 1。合并 *Weather*，其中 $P(\textit{Weather} = \textit{norain}) = 0.7$ ，得到的期望效用为 $0.7 \times 100 + 0.3 \times 70 = 91$ 。

9.4 信息与控制的价值

【例 9-20】 在例 9-18 中，检查烟雾的动作 *CheckSmoke* 为火情提供了信息。检查烟雾的成本是 20，并且没有直接的回报。然而，在一个最优策略中，当有报告人们离开建筑物时做这个检查是值得的，因为 Agent 可以根据检查获得的信息做出下一步的动作。因此，有关烟雾的信息对于 Agent 来说是很有价值的。尽管有关烟雾的信息无法提供是否有火情的完美信息，但该信息对于作出决策仍然非常有用。

396

从这个例子得到的一个重要启发是搜集信息的动作，正如检查烟雾的动作 *check_for_*

smoke 一样, 可以同样的方式用在其他任何动作上, 如打电话给消防部门 *call_fire_department*。一个最优的策略常常包含以搜集信息为唯一目的的动作, 只要后续的动作依赖于该动作产生的一些影响。大多数的动作都不会仅仅提供信息, 而是更多地对所处环境产生直接的影响。

对于 Agent 来说, 信息是很有价值的, 因为它可帮助 Agent 做出更好的决策。

信息 i 对于决策 D 的信息价值 (value of information) 是一个依赖于 D 及其后续决策的最优策略在获得信息 i 时的期望效用减去不能观察到 i 的最优策略的期望效用。因此, 在决策网络中, 信息价值就是一个含有以 i 为 D 及其后续策略的父节点之一的最优策略的效用减去一个不含有以 i 为 D 的父节点的最优策略的效用。

【例 9-21】 在例 9-11 中, 考虑获得一个更好的天气预报的价值。为决定是否带伞, 获得关于天气的完全信息的价值等于在网络中增加一条从 *Weather* 到 *Umbrella* 的边时的最优策略期望效用 (在例 9-19 中计算, 其结果为 91) 减去原始决策网络最优策略的期望效用 (在例 9-11 中计算, 其结果为 77)。因此, 关于天气的完全信息的价值是 $91 - 77 = 14$ 。这是另一个感知天气情况的传感器的价值上限。

信息价值限定了 Agent 为获得支持决策 d 的信息 i 而愿付出的代价 (就效用的损失而言)。该数值是有关决策 d 的信息 i 的不完全信息的价值上限。不完全的信息是指类似于从一个有噪声的传感器获得的信息 i 。为获得信息 i 而付出的成本不应超过信息的价值。

信息的价值有如下一些有意思的属性:

- 信息价值不会是负值。最坏的情况是 Agent 忽略这个信息。
- 如果一个最优决策总是做同一个动作, 而与信息 i 的取值无关, 则该信息 i 的价值为 0。如果信息 i 的价值为 0, 则存在一个不依赖于信息 i 的最优策略 (即不管信息 i 取何值, Agent 总是选择同一个动作)。

在一个决策网络中, 作出决策 d 时的信息 i 的价值可以通过考虑下面两个决策网络来计算:

- 决策网络中从 i 到 d 的边及从 i 到后续决策的边。
- 决策网络中没有这样的边。

上述两个决策网络的最优策略的期望效用之差即为作出决策 d 时信息 i 的价值。当加入从 i 到 d 的边会造成环时, 需要先做一些处理。

【例 9-22】 在报警问题的例子中 (例 9-18), Agent 可能想知道安装一个中继报警设备——从而可以直接听到报警声而不是依赖于人们离开建筑物的嘈杂声——是否值得。为确定该中继报警设备的价值, 需要考虑关于报警的完全信息的价值。如果信息价值低于中继报警设备的成本, 则就不值得安装该设备了。

对于检查烟雾和打电话给消防部门的决策, 关于报警 *Alarm* 的信息的价值可这样计算: 求解图 9-9 的决策网络, 及求解同一个决策网络但包含了从报警 *Alarm* 到检查烟雾 *Check_for_smoke* 的边和从 *Alarm* 到打电话给消防部门 *Call_fire_department* 的边。原始的决策网络的期望效用是 -22.6。新的决策网络中有一个期望效用为 -6.3 的最优策略。两个决策网络的最优策略的期望效用之差为 16.3——这就是 *Alarm* 对于决策 *Check_for_smoke* 的价值。如果中继报警设备的成本为 20, 则不值得安装该设备。

含有从 *Alarm* 到 *Call_fire_department* 的边的决策网络的期望效用为 -6.3, 与含有从 *Alarm* 到 *Check_for_smoke* 的边时的期望效用相同。在最优策略中, *Check_for_smoke* 的最优决策函数会忽略关于 *Alarm* 的信息, 即当 *Alarm* 是 *Call_fire_department* 的父节

点时，根据决策网络的最优策略，Agent 从不会检查烟雾。◀

控制价值(value of control)明确说明控制一个变量的价值的多少。其最简单的形式是将一个决策网络的随机变量替换成决策变量，并增加相应的边构成无遗忘网络时，其最优策略的期望效用的变化量。随机变量替换为决策变量后，期望效用的变化量不会是负值；新构成的决策网络总有相等或更高的期望效用。

【例 9-23】 在图 9-9 的报警决策网络中，你可能会对控制意外干扰感兴趣。为此可估算为增加安全性而防止意外干扰的价值。为计算这一点，可比较图 9-9 中的决策网络的期望效用与将随机变量意外干扰 *Tampering* 替换为决策变量(同时也成为另外两个决策节点的父节点)后的期望效用。

初始的决策网络的期望效用是 -22.6。首先考虑该变量的价值。如果 *Tampering* 作为 *Call* 的父节点，则新的期望效用为 -21.30。如果 *Tampering* 作为 *Call* 和 *CheckSmoke* 的父节点，则新的期望效用为 -20.87。

为确定该变量的控制价值，先将 *Tampering* 节点改为决策节点并将它作为另外两个决策节点的父节点。新构成的决策网络的期望效用为 -20.71。请注意，在这里控制价值比信息价值高。

在原始的决策网络中，意外干扰的控制价值为 $-20.71 - (-22.6) = 1.89$ 。在观察意外干扰的情况下，意外干扰的控制价值为 $-20.71 - (-20.87) = 0.16$ 。◀

398

上述描述适用于随机变量的父节点在该随机变量变成决策变量后成为决策变量的父节点的情况。在这种情形下，控制价值从不会是负值。然而，如果决策变量的父节点没有包括原随机变量的所有父节点，则控制价值有可能少于信息价值。一般来说，在考虑将一个随机节点转换成决策节点时，必须明确决策节点可用的信息。

【例 9-24】 考虑将图 9-9 中的变量 *Smoke* 变成决策变量的情况。如果火情 *Fire* 是决策变量 *Smoke* 的父节点，则它必然是 *Call* 的一个父节点，从而构成一个无遗忘网络。在 *Smoke* 决策早于决策 *checkSmoke* 时，生成的网络的期望效用为 -2.0。在这种情况下，将 *Smoke* 转换为决策变量的原因是对 *Fire* 的观察。此时，最优的策略是如果有火情则打电话给消防部门，否则就不打电话。

假设将 *Smoke* 转换为决策变量，且 *Fire* 不再作为该决策的条件。也就是说，Agent 不得不自行决定是否烟雾，且 *Fire* 也不作为其他决策的条件。这种情况可以建模为 *Smoke* 是决策变量且没有父节点。此时，网络的期望效用是 -23.20——劣于初始的决策网络，其原因是盲目地控制 *Smoke* 造成失去了感知 *Fire* 的能力。◀

9.5 决策过程

9.4 节中的决策网络具有有限个阶段和部分可观察的范围。本节讨论具有无限定时域和无穷时域的问题。

通常，Agent 必须为一个持续的过程进行推理，即它不知道需要推理多少步。如果这个过程永远持续下去，则称该类问题为**无穷时域**(infinite horizon)问题；如果 Agent 最终会停止推理，但不知道何时会停止推理，则称该类问题为**无限定时域**(indefinite horizon)问题。为了给这种情况建模，我们扩展马尔可夫链以处理 Agent 的行动。在每一个阶段，Agent 决定执行哪个问题；结果状态取决于前一个状态和执行的动作。

对于持续的过程，你不会只想考虑最后的效用，因为 Agent 的推理有可能永远也不会结束。相反，Agent 会收到一系列的**回报**(reward)。这些回报涵盖了执行动作的成本，及

可能接受的奖励或惩罚。负回报被称为惩罚(punishment)。无限定时域问题可以使用一个停止态来建模。**停止态(stopping state)**或**吸收态(absorbing state)**是所有的动作都不会产生任何影响的状态,也就是说,当 Agent 进入该状态后,所有的动作都会直接返回,并保持状态不变,且回报为零。目标的成绩可建模成 Agent 进入停止态时所获得的回报。

我们只考虑平稳的(stationary)模型,即其中的状态转换和回报大小都不依赖于时间。

一个马尔可夫决策过程(Markov decision process, MDP)的组成如下:

- S , 各种情景中的状态集合。
- A , 动作集合。
- $P: S \times S \times A \rightarrow [0, 1]$, 用来描述动态性,并记为 $P(s' | s, a)$ 且有

$$\forall s \in S \forall a \in A \sum_{s' \in S} P(s' | s, a) = 1$$

特别地, $P(s' | s, a)$ 定义了 Agent 在状态 s 下执行动作 a 时转向状态 s' 的概率。

- $R: S \times A \times S \rightarrow \mathbb{R}$, 其中 $R(s, a, s')$ 定义了执行动作 a 并从状态 s 转向状态 s' 时的期望回报。

动态性和回报都可以是随机的,即转移的状态和回报可以有一定的随机性,这通过在转移状态上设置概率分配函数和通过 R 确定期望回报来实现。当转移状态和回报依赖于尚未在 MDP 模型中的随机变量时,它们也是随机的。

图 9-12 的决策网络描述了一个马尔可夫决策过程的部分内容。

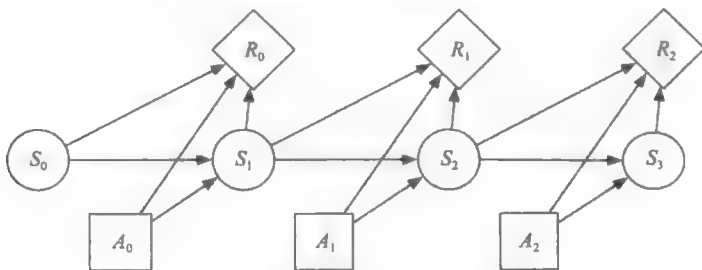


图 9-12 表示部分 MDP 的决策网络

【例 9-25】 网格化的世界是机器人的一个理想环境。任何时刻,机器人在某个位置并能移向近邻的位置,并为此获得回报或者接受惩罚。假设机器人的动作是随机的,也就是说结果状态服从一个概率分布,且此概率依赖于执行的动作和所处的状态。

图 9-13 是一个 10×10 的网格世界,在这个世界中机器人可以执行 4 个动作:向上、向下、向左、向右。如果机器人执行了其中一个动作,则该动作有 0.7 的概率朝向想要去的方向,分别有 0.1 的概率朝向另外三个方向。如果碰到外壁(即计算出的位置在给定的网格之外),则需要接受 1 个单位的惩罚(即回报为 -1)且并不真正移动位置。共有 4 个奖励状态(除去墙壁):一个奖励为 +10 的位置(在位置 (9, 8) 处,第 8 行第 9 列),一个奖励为 +3 的位置(在位置 (8, 3) 处),一个奖励为 -5 的位置(在位置 (4, 5) 处),及一个奖励为 -10 的位置(在位置 (4, 8) 处)。在其中的每一个状态,Agent 在该状态执行

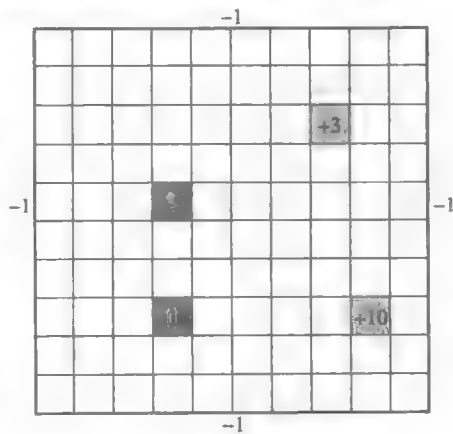


图 9-13 例 9-25 使用的网格世界

了动作后才获得奖励，而不是进入该状态时就获得奖励。当 Agent 到达状态(9, 8)时，不管它下一步做什么，它都会在网格世界的四个角中任选一个并移过去。

请注意，在这个例子中，回报是初始状态和最终状态的函数。判断 Agent 是否撞到了墙上的方法是检查 Agent 是否发生了实际的移动。仅知道 Agent 的初始状态和动作，或者仅知道最终状态和动作，都不足以提供推断回报所需的信息。

对于 9.3.1 节的决策网络，设计者不得不考虑在 Agent 作决策时能够利用哪些信息。这里有两种常见的变化：

- 在一个完全可观察马尔可夫决策过程(fully observable Markov decision process)中，Agent 在决策要做什么动作时开始观察当前的状态。
- 一个部分可观察马尔可夫决策过程(partially observable Markov decision process)是一个 MDP 和一个隐马尔可夫模型的组合。在每一个时间点，Agent 开始根据状态做一些观察。Agent 在作决策时可以访问观察和之前动作的历史情况。它不能直接观察当前的状态。

为决定要做什么，Agent 要比较不同的回报序列。完成这个比较的最常见方法是将回报序列转换成称为值(value)或累积回报(cumulative reward)的数值。为此目的，Agent 将直接回报与将来的其他回报组合在一起。假设 Agent 接收到的回报序列如下：

$$r_1, r_2, r_3, r_4, \dots$$

则有如下三种常见的将回报组合成值 V 的方法：

总回报(total reward): $V = \sum_{i=1}^{\infty} r_i$ 。此时，值为所有回报之和。如果你能保证回报之和是有限的，则这种办法就是适用的；但如果回报之和是无限的，则无法比较哪个回报序列更好。例如，一个回报均为 1 美元的无限序列之和与回报均为 100 美元的序列之和相等(两者均为无穷大)。此时，回报之和变为有限的情况是存在一个终止条件；当 Agent 总是能以一个大于零的概率进入停止状态时，回报之和就是有限的了。

平均回报(average reward): $V = \lim_{n \rightarrow \infty} (r_1 + \dots + r_n)/n$ 。此时，Agent 的值评价是其各时刻所得回报的平均值。只要回报是有限的，则该评价就是有限的。不过，如果总回报是有限的，则平均回报就变成了 0，从而 Agent 就无法根据此数值来确定该执行哪个动作，因为无论执行哪个动作，平均回报都是 0。在这个标准下，唯一重要的事情是 Agent 会在哪里停止。任何有限的不佳动作序列都不会影响最终的极限数值。例如，在获得 100 万美元的回报之后每次仅获得 1 美元的回报，与获得 0 美元的回报之后每次都获得 1 美元的回报，其平均回报是相等的(它们的平均回报的极限均为 1 美元)。

折扣回报(discounted reward): $V = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots + \gamma^{t-1} r_t + \dots$ ，其中 γ 是折扣因子(discount factor)， $0 \leq \gamma < 1$ 。在这个标准下，将来回报的重要程度要低于当前的回报。如果 γ 等于 1，则该标准与总回报等价。当 $\gamma=0$ 时，Agent 忽略将来的回报。无论总回报是否是有限的， $0 \leq \gamma < 1$ 就保证了该标准总是有限的。

折扣回报可写为：

$$\begin{aligned} V &= \sum_{i=1}^{\infty} \gamma^{i-1} r_i \\ &= r_1 + \gamma r_2 + \gamma^2 r_3 + \dots + \gamma^{t-1} r_t + \dots \\ &= r_1 + \gamma(r_2 + \gamma(r_3 + \dots)) \end{aligned}$$

假设 V_k 是从时刻 k 开始累积回报，则有：

$$V_k = r_k + \gamma(r_{k+1} + \gamma(r_{k+2} + \dots))$$

$$= r_k + \gamma V_{k+1}$$

为理解 V_k 的性质, 假设 $S = 1 + \gamma + \gamma^2 + \gamma^3 + \dots$, 则有 $S = 1 + \gamma S$ 。求解 S 得 $S = 1/(1-\gamma)$ 。因此, 在折扣回报的标准下, 将来的全部值至多是 $1/(1-\gamma)$ 与最大单次回报的乘积, 至少是 $1/(1-\gamma)$ 与最小单次回报的乘积。因此, 相比立即得到的回报, 从现在开始的无限时间内的值是有限的。这与平均回报不同。在平均回报的标准下, 立即回报被无限时间的累积回报所支配。

402

在经济学中, γ 与利率相关: 现在得到 1 美元等价于一年后得到 $1+i$ 美元, 其中 i 为利率。你可以把折扣率看做 Agent 的生存概率; γ 可以看成是 Agent 继续行动下去的概率。本书后面部分考虑的是折扣回报。

一个平稳策略(stationary policy)是一个函数 $\pi: S \rightarrow A$, 即该函数给每一个状态指定一个动作。给定一个回报标准后, 一个策略可为每一个状态计算出期望值。设 $V^\pi(s)$ 是在状态 s 下由 π 给出的期望值。这反映了 Agent 在该状态下按照策略 π 行动时获得的期望值。如果不存在一个策略 π' 及状态 s 满足 $V^{\pi'}(s) > V^\pi(s)$, 则称策略 π 为最优策略。也就是说, 对于每一个状态, 该策略比其他策略具有更高或相等的期望值。

对于无穷时域问题, 一个平稳的 MDP 总有一个最优的平稳策略。然而, 对于有限阶段问题却不是如此——在该问题中, 一个非平稳策略可能要好于所有的平稳策略。例如, 如果 Agent 不得不在时刻 n 时停止, 在某个状态下做出最后一个决定, 则 Agent 可能执行能够获得最大的立即回报的动作而不再考虑将来的动作, 但在之前处于同一状态时的决策中, 它可能会决定执行一个获得较低立即回报但有望随后获得较大回报的动作。

9.5.1 策略值

对于折扣率为 γ 的折扣回报, 策略 π 的期望值由两个相互关联的函数 V^π 和 Q^π 定义。

设 $Q^\pi(s, a)$ 是在状态 s 下根据策略 π 执行动作 a 时的期望值。回想一下, $V^\pi(s)$ 是在状态 s 下遵循策略 π 的期望值。

Q^π 和 V^π 彼此可以递归定义。如果 Agent 处于状态 s , 执行动作 a 并到达状态 s' , 它就得到立即回报 $R(s, a, s')$ 及将来的折扣回报 $\gamma V^\pi(s')$ 。在 Agent 做规划时, 它并不知道确切的结果状态, 因此它使用期望值, 即所有结果状态的值的平均值:

$$Q^\pi(s, a) = \sum_{s'} P(s' | s, a) (R(s, a, s') + \gamma V^\pi(s')) \quad (9.2)$$

其中 $V^\pi(s)$ 可通过执行策略 π 确定的动作并按照该策略执行得到:

$$V^\pi(s) = Q^\pi(s, \pi(s))$$

403

9.5.2 最优策略值

设 $Q^*(s, a)$ 是 Agent 在状态 s 下执行动作 a 并遵循最优策略的期望值。设 $V^*(s)$ 是从状态 s 开始遵循最优策略的期望值。

Q^* 的定义类似于 Q^π 的定义:

$$Q^*(s, a) = \sum_{s'} P(s' | s, a) (R(s, a, s') + \gamma V^*(s')) \quad (9.3)$$

其中 $V^*(s)$ 是在每个状态中执行使得值最大的动作所得的值:

$$V^*(s) = \max_a Q^*(s, a)$$

最优策略 π^* 是使得 Agent 在每一个状态下均能获得最大值的策略:

$$\pi^*(s) = \arg \max_a Q^*(s, a)$$

请注意, $\arg \max_a Q^*(s, a)$ 是状态 s 的一个函数, 其值为使 $Q^*(s, a)$ 最大的动作 a 。

9.5.3 值迭代

值迭代是一个计算最优 MDP 策略及其值的方法。

值迭代从一个“终点”开始, 然后不断向后计算, 逐步精化 Q^* 或 V^* 的估计值。算法没有真正的终点, 因此它需要有一个人为指定的终点。设 V_k 为阶段 k 的值函数, Q_k 为阶段 k 的 Q 函数。它们可以递归定义。值迭代从一个任意的函数 V_0 开始, 然后使用下面的方程式从 k 阶段的函数得到 $k+1$ 阶段的函数:

$$Q_{k+1}(s, a) = \sum_{s'} P(s' | s, a) (R(s, a, s') + \gamma V_k(s')) \quad k \geq 0$$

$$V_{k+1}(s) = \max_a Q_{k+1}(s, a) \quad k \geq 0$$

计算过程中可选择存储 $V[S]$ 数组, 也可选择存储 $Q[S, A]$ 数组。存储 V 数组需要的空间少, 但确定一个最优动作比较困难, 并且需要更多一次迭代来确定哪个动作的价值最大。

图 9-14 显示了存储 V 数组的值迭代算法。无论初始值函数 V_0 的取值为何, 该过程都是收敛的。接近 V^* 的初始值函数比远离 V^* 的初始值函数的收敛速度快。许多 MDP 的提取技术的基础是使用一些启发式方法来近似 V^* 并将该近似值作为值迭代的初始种子。

404

```

1: procedure Value_Iteration(S, A, P, R,  $\theta$ )
2:   Inputs
3:   S: 所有状态的集合
4:   A: 所有动作的集合
5:   P: 状态转换函数  $P(s' | s, a)$ 
6:   R: 回报函数  $R(s, a, s')$ 
7:    $\theta$ : 一个阈值,  $\theta > 0$ 
8:   Output
9:    $\pi[S]$ : 近似最优策略
10:  V[S]: 值函数
11:  Local
12:  实数取值的数组  $V_i[S]$ , 一个值函数序列
13:  动作数组  $\pi[S]$ 
14:  为  $V_0[S]$  任意赋值
15:   $k := 0$ 
16:  repeat
17:     $k := k + 1$ 
18:    for each 状态  $s$  do
19:       $V_k[s] = \max_a \sum_{s'} P(s' | s, a) (R(s, a, s') + \gamma V_{k-1}[s'])$ 
20:  until  $\forall s | V_k[s] - V_{k-1}[s] | < \theta$ 
21:  for each 状态  $s$  do
22:     $\pi[s] = \arg \max_a \sum_{s'} P(s' | s, a) (R(s, a, s') + \gamma V_k[s'])$ 
23:  return  $\pi, V_k$ 

```

图 9-14 MDP 的值迭代算法, 存储数组 V

【例 9-26】 考虑例 9-25 中回报为 +10 的格子及其周围的格子。折扣为 $\gamma = 0.9$ 。对于所有的状态 s , 假设算法的初始值函数为 $V_0[s] = 0$ 。

对于上述9个格子，假设值函数 V_1 、 V_2 和 V_3 的取值如下：

0	0	-0.1	0	6.3	-0.1	4.5	6.2	4.4
0	10	-0.1	6.3	9.8	6.2	6.2	9.7	6.6
0	0	-0.1	0	6.3	-0.1	4.5	6.1	4.4

在第一次值迭代后，节点会得到直接预期回报。该图的中间节点是回报为+10的状态。右边节点的价值为-0.1，从而其最优动作为向上、向左和向下；这些动作都有0.1的概率撞到墙上去，此时的回报为-1。

中间的网格表示了 V_2 ，即第二步值迭代后的函数值。考虑+10回报状态左边直接相邻的节点：其最优值为向右；它有0.7的概率在下一个状态得到10的回报，因此其价值为9(10乘折扣率0.9)；其他可能状态的期望回报为0。因此，上述状态的价值为 $0.7 \times 9 = 6.3$ 。

405

考虑第二步值迭代后+10回报状态的右边直接相邻的节点。Agent在此状态时的最优动作是向左，该状态的价值为：

	概率	回报		未来价值	
	$0.7 \times ($	0	+	$0.9 \times 10)$	Agent 向左
+	$0.1 \times ($	0	+	0.9×-0.1	Agent 向上
+	$0.1 \times ($	-1	+	$0.9 \times -0.1)$	Agent 向右
+	$0.1 \times ($	0	+	$0.9 \times -0.1)$	Agent 向下

最后得到的值为6.173。

请注意+10回报状态的值是如何小于10的。这是因为Agent冲向了某个角落，而这些角落在此时是不佳的状态。

再做一次值迭代(见上图的右边)，+10回报状态的影响更进一步。特别的，角落状态得到的值表示了3次迭代后的回报。

本书网站上有一个 applet 小程序演示了本例值迭代的详细过程。

图9-14中的值迭代算法为每一个阶段设置了一个数组，但实际上仅需要保存当前数组和前一个数组。算法可轮换地根据一个数组的值更新另一个数组的值。

该算法的一个常见改进是**异步值迭代**(asynchronous value iteration)。与计算完所有状态然后再创建一个新的值函数的做法不同，异步值迭代在计算了一个状态后立即进行更新(任意顺序都可以)，并将这些值保存在一个数组中。异步值迭代可以保存 $Q[s, a]$ 数组，也可以保存 $V[s]$ 数组。图9-15给出了异步值迭代算法过程，其中保存了 Q 数组。该算法比值迭代算法的收敛速度更快、占用空间更少，是一些强化学习算法的基础。如果Agent必须保证一定的误差范围，则算法的终止条件就很难确定，除非仔细考虑动作和状态的选择。通常情况下，这个过程无限期地运行，同时做好回答关于最优动作的最佳估计的询问。

异步值迭代也可以通过仅存储 $V[s]$ 数组来实现。此时，算法选择一个状态 s 并通过下式更新：

$$V[s] = \max_a \sum_{s'} P(s' | s, a) (R(s, a, s') + \gamma V[s'])$$

尽管这个算法存储了较少的信息，但要抽取出策略却更加困难。它需要一个额外的备份，以确定哪个动作 a 产生的价值是最大的。这可用下式实现：

$$\pi[s] = \arg \max_a \sum_{s'} P(s' | s, a) (R(s, a, s') + \gamma V[s'])$$

406

```

1: procedure Asynchronous_Value_Iteration(S, A, P, R)
2:   Inputs
3:     S: 所有状态的集合
4:     A: 所有动作的集合
5:     P: 状态转换函数  $P(s' | s, a)$ 
6:     R: 回报函数  $R(s, a, s')$ 
7:   Output
8:      $\pi[s]$ : 近似最优策略
9:      $Q[S, A]$ : 值函数
10:  Local
11:    实数数组  $Q[S, A]$ 
12:    动作数组  $\pi[S]$ 
13:    为  $Q[S, A]$  赋任意的初始值
14:  repeat
15:    选择一个状态  $s$ 
16:    选择一个动作  $a$ 
17:     $Q[s, a] = \sum_i P(s' | s, a) (R(s, a, s') + \gamma \max_{a'} Q[s', a'])$ 
18:  until 结束条件
19:  for each 状态  $s$  do
20:     $\pi[s] = \arg \max_a Q[s, a]$ 
21:  return  $\pi, Q$ 

```

图 9-15 MDP 的异步值迭代算法

【例 9-27】在例 9-26 中，从 +10 回报状态向上一步再向左一步的状态的值是在进行了三次值迭代之后才更新的。在上述三次值迭代中，所有的状态都进行了计算。

在异步值迭代中，可以首先选择 +10 回报状态。然后可选择向左一步，并可计算出其值为 $0.7 \times 0.9 \times 10 = 6.3$ 。然后可选择该节点上面的那个节点，并可计算其值为 $0.7 \times 0.9 \times 6.3 = 3.969$ 。请注意，在考虑了 3 个状态（而不是 300 个状态，就像值迭代中那样）后得到的这个值就反映了它接近于一个 +10 回报状态。

9.5.4 策略迭代

策略迭代(policy iteration)从一个策略开始，然后迭代改进它。算法从任意一个策略 π_0 (越接近最优策略越好)开始，然后执行下面的步骤，其中 i 从 0 开始。

- 策略评估：确定 $V^{\pi_i}(S)$ 。 V^* 由一组共 $|S|$ 个线性方程定义，其中有 $|S|$ 个未知变量。这些未知变量就是 $V^{\pi_i}(S)$ 的取值。每一个状态对应一个方程。这个线性方程组可用解线性方程组的方法(如高斯消去法)求解，或者可以用迭代的方式求解。
- 策略改进：选择 $\pi_{i+1}(s) = \arg \max_a Q^{\pi_i}(s, a)$ ，其中 Q 值可使用式(9.2)根据 V 的值计算出来。为检测算法何时已经收敛，应该仅在某些状态下的新动作改进了期望值时才改变策略，也就是说，如果 $\pi_i(s)$ 是一个使 $Q^{\pi_i}(s, a)$ 最大的动作，则应该将 $\pi_{i+1}(s)$ 设为 $\pi_i(s)$ 。
- 当策略不再变动时停止，即 $\pi_{i+1} = \pi_i$ 时停止；否则， i 增 1 并重复本过程。

算法如图 9-16 所示。请注意，该算法仅保留了最新的策略，并知道它是否已被改变。该算法总会停止，且通常不会迭代很多步。不幸的是，解线性方程组通常很耗时。

一个策略迭代算法的改进，称为改进的策略迭代(modified policy iteration)，是利用算法并不要求 Agent 评估策略这个特点而得到的；算法可以备份多步行动(使用式(9.2))，然后再做改进。

```

1: procedure Policy_Iteration( $S, A, P, R$ )
2:   Inputs
3:      $S$ : 所有状态集合
4:      $A$ : 所有动作集合
5:      $P$ : 状态转移函数  $P(s' | s, a)$ 
6:      $R$ : 回报函数  $R(s, a, s')$ 
7:   Output
8:     最优策略  $\pi$ 
9:   Local
10:    动作数组  $\pi[S]$ 
11:    布尔变量  $noChange$ 
12:    实数数组  $V[S]$ 
13:    设置任意一个策略  $\pi$ 
14:    repeat
15:       $noChange \leftarrow true$ 
16:      求解  $V[s] = \sum_{s' \in S} P(s' | s, \pi[s]) (R(s, a, s') + \gamma V[s'])$ 
17:      for each  $s \in S$  do
18:        设  $QBest = V[s]$ 
19:        for each  $a \in A$  do
20:          Let  $Qsa = \sum_{s' \in S} P(s' | s, a) (R(s, a, s') + \gamma V[s'])$ 
21:          if  $Qsa > QBest$  then
22:             $\pi[s] \leftarrow a$ 
23:             $QBest \leftarrow Qsa$ 
24:             $noChange \leftarrow false$ 
25:    until  $noChange$ 
26:    return  $\pi$ 

```

图 9-16 MDP 的策略迭代算法

408

对于过于庞大而难以直接用 MDP 表示的系统，策略迭代所蕴含的思想是可供借鉴的。假设一个控制器有一些参数，这些参数的取值能够变化。对一些环境 s 中的参数 a 的累积折扣回报的导数的估计（对应于 $Q(a, s)$ 的导数），能够用来改进参数。这样迭代改进的控制器能够陷入一个非全局最大的局部极大。MDP 的策略迭代不会停止在非最优的局部极大，因为在那种状态下，可以改进一个状态下的动作而不影响其他的状态，然而，对参数的更新能够一下子影响许多状态。

9.5.5 动态决策网络

MDP 是基于状态的表示。在本节，我们考虑一个基于特征的 MDP 扩展，它是决策理论规划 (decision theoretic planning) 的基础。

动态决策网络 (dynamic decision network, DDN) 的表示有多种形式：

- MDP 的因子化表示：其中的状态描述为特征；
- 决策网络的扩展：允许重复进行中的动作和状态的变化；
- 动态信念网络的扩展：包含了动作和回报；
- 动作的基于特征的表示的扩展：允许动作的影响具有不确定性。

一个完全可观察的动态决策网络的组成如下：

- 一个状态特征的集合，每一个特征有一个值域；
- 由一个可能动作的集合构成的决策节点 A ，决策的值域即为动作集合；

- 一个有一个动作节点 A 、对应于每一个特征 F 的节点 F_0 和 F_1 (分别表示时刻 0 和时刻 1 时的特征), 及一个条件概率 $P(F_1 | \text{parents}(F_1))$ 的两阶段信念网络, 其中 F_1 的双亲节点可以包括节点 A 和时刻 0、时刻 1 的特征, 只要所得的网络是无环的;
- 一个回报函数, 它是动作和时刻 0 与时刻 1 的任意特征的函数。

409

正如动态信念网络中的那样, 时刻 1 的特征能被复制作为后续时刻的特征。

【例 9-28】 考虑将例 8-1 的一个随机版本表示为一个动态决策网络。我们使用与例 8-1 中相同的特征。

$RLoc_1$ 的双亲为 $RLoc_0$ 和 A 。 RHC_1 的双亲为 RHC_0 、 A 和 $RLoc_0$; 机器人是否有咖啡取决于它之前是否有咖啡、它做过什么动作以及它的位置。

SWC_1 的双亲包括 SWC_0 、 RHC_0 、 A 和 $RLoc_0$ 。你不会认为 RHC_1 和 SWC_1 是独立的, 因为它们都依赖于咖啡是否被成功送到。这可以通过将其中一个设为另一个的双亲节点来表示。在时刻 1, 两阶段信念网络的状态变量依赖于选择的动作及图 9-17 中的其他状态变量。该图也表示了回报是动作的函数, 该动作可以是 Sam 不再想要咖啡, 也可以是等待取信。

另外一个建模 RHC_1 和 SWC_1 间的依赖关系的方法是引入一个新的表示咖啡是否在时刻 1 成功送到的变量 CSD_1 。该变量是 RHC_1 和 SWC_1 的双亲。 Sam 是否想要咖啡是 Sam 之前是否想要咖啡及咖啡是否成功送到的函数。机器人是否有咖啡取决于其动作和位置, 以此来建模机器人端起咖啡这件事。相似地, MW_1 与 RHM_1 间的依赖关系可通过引入一个表示是否成功拿起信件的变量 MPU_1 来建模。其结果对应为图 9-18 中的有 2 层的 DDN, 但略去了回报。

410

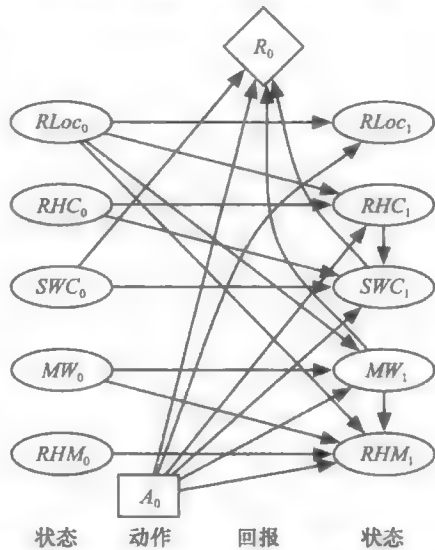


图 9-17 表示了两阶段信念网络和回报结构的动态决策网络

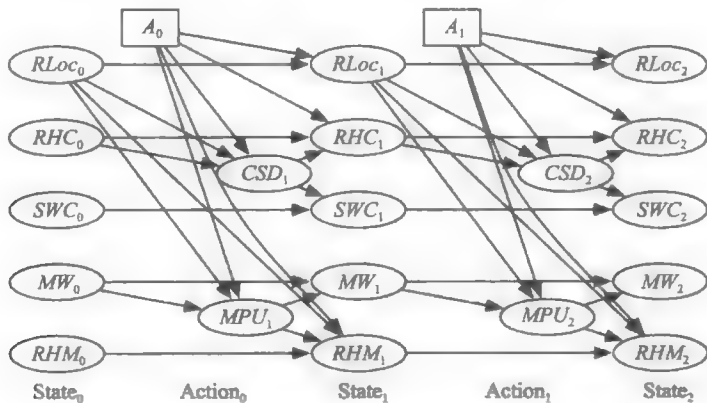


图 9-18 有 2 层结构的带有中间变量的动态决策网络, 略去了回报节点

作为此类决策网络的一部分, 我们也应该为关于动作和回报的可用信息建模。在一个完全可观察动态决策网络中, 动作的双亲都是之前的状态变量。因为这一点可通过推理得

到, 因此相应的边通常并不绘出。如果回报仅出现在最后, 就能直接应用决策网络的变量消除法(如图 9-11 所示)。请注意, 对此工作, 我们不要求无遗忘条件; 有了完全可观察条件就足够了。如果在每一步都要收集回报, 必须对算法进行扩充以适应回报的累加。参考习题 9.12。

9.5.6 部分可观察决策过程

部分可观察马尔可夫决策过程(POMDP)是 MDP 和隐马尔可夫模型的组合。其中不再假设状态是可观察的, 而是假设 Agent 在必须行动之前仅可得到部分和/或有噪声的观察。

一个 POMDP 由以下部分组成:

- S : 环境中的状态集合。
- A : 动作集合。
- O : 可能得到的观察的集合。
- $P(S_0)$: 初始状态的概率分布。
- $P(S' | S, A)$: 指定了动态性, 即在状态 S 下执行动作 A 从而转换到 S' 的概率。
- $R(S, A, S')$: 在状态 S 下执行动作 A 并转换到状态 S' 的期望回报。
- $P(O | S)$: 在状态 S 下能够观察到 O 的概率。

POMDP 的有限部分可用决策图表示, 如

图 9-19 所示。

计算 POMDP 的最优策略的方法主要有三种:

- 使用决策网络的变量消除法来求解动态决策网络(如图 9-11, 需要扩展以处理折扣回报)。创建的策略是 Agent 历史的一个函数。使用此方法的问题是 Agent 的历史是无界的, 且可能的历史数量在规划过程中是以指数增长的。
- 策略是信念状态的一个函数, 即是各种状态上的概率分布。维护信念状态是一个滤波问题。该方法的问题是: 对于 n 个状态, 信念状态集合是一个 $(n-1)$ 维的实数空间。然而, 因为一系列动作的值仅依赖于状态, 所以期望值是状态值的一个线性函数。因为规划以观察为条件, 且对任意信念状态我们仅考虑其最优动作, 所以任意有限前瞻的最优策略是分段线性和凸的。
- 搜索控制器的空间找到最好的控制器。由此, Agent 根据其信念状态和观察搜索要记忆哪些、要怎么做。请注意, 前两个方法是此方法的两个实例: Agent 记忆了其所有的历史, 或 Agent 有一个信念状态且该状态是所有可能状态的概率分布。通常, Agent 希望记忆其一部分的历史, 但对另外一些特征有概率选择的能力。因为对要记忆哪些没有加以限制, 所以搜索空间是巨大的。

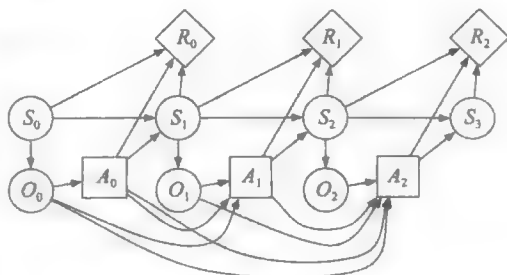


图 9-19 POMDP 的动态决策网络表示

9.6 本章小结

- 效用是衡量偏好的指标, 而偏好包含有不确定性。
- 决策网络可以使用特征来表示一个有限阶段的部分可观察顺序决策问题。

- MDP 可以使用状态来表示无穷阶段或无限定阶段顺序决策问题。
- 完全可观察 MDP 能够使用值迭代或策略迭代算法求解。
- 动态决策网络可使用特征来表示 MDP。

9.7 参考文献及进一步阅读

这里介绍的效用理论由 Neumann 和 Morgenstern[1953]提出,并由 Savage[1972]做了改进。Keeney 和 Raiffa[1976]讨论了效用理论,并集中于多属性(基于特征)效用函数的讨论上。对于效用的图形化模型,请参考 Bacchus 和 Grove[1995]的工作,及 Boutilier、Brafman、Domshlak、Hoos 和 Poole[2004]的工作。近期, Walsh[2007]对相关工作做了一个总结。

决策网络或影响图由 Howard 和 Matheson[1984]提出。使用动态规划求解影响图的方法可参考 Shachter 和 Peot[1992]的工作。Matheson[1990]讨论了信息和控制的价值。

MDP 由 Bellman[1957]提出,Puterman[1994]和 Bertsekas[1995]也对此做了讨论。Boutilier、Dean 和 Hanks[1999]总结了将 MDP 逐步升华为决策理论规划的过程。

9.8 习题

9.1 学生必须对每门课的学习程度作出决策。这个问题的目的是研究如何使用决策网络,以帮助他们做出这样的决定。

假设学生首先要决定为了期中考试要学习多少。他们可以选择学习很多,学习一点儿,或者不学习。他们是否通过期中考试取决于他们学习的多少及课程的难度。作为一阶近似,如果他们努力学习,或课程简单且学习了一点儿,则他们能够通过期中考试。在得到期中考试的分数的之后,他们必须决定为了期末考试要做出怎样的努力。与之前一样,期末的成绩与他们的努力程度和课程的难易有关。他们的最后分数取决于他们通过了哪些考试。一般地,如果他们通过了两次考试,则分数为 A;如果仅通过了期末考试,则得分为 B;如果仅通过期中考试,则得分为 C;如果两次考试都没有通过,则得分为 F。当然,在这个评分标准中存在着大量的噪声。

假设他们最后的效用依赖于他们学习的总付出和他们最后的成绩。假设学习总付出是为期中考试而做出的学习付出加上为期末考试而做出的学习付出。

(a) 根据以上描述画出学生作决策的决策网络。

(b) 每一个变量的值域是什么?

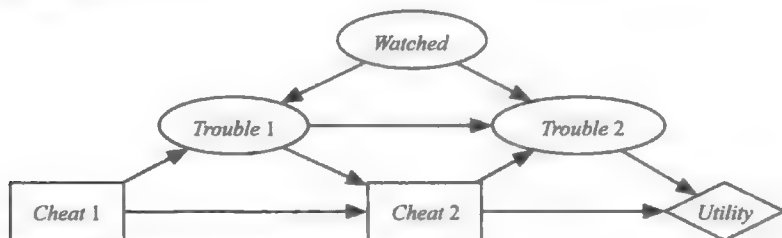
(c) 给出一个合理的条件概率表。

(d) 最优结果是什么(此时的效用为 100)? 最差的结果是什么(此时的效用为 0)?

(e) 为一个只想通过考试(不得 F)的学生设计一个合适的效用函数。该学生的最优策略是什么?

(f) 为一个能做到很好的学生设计一个合适的效用函数。该学生的最优策略是什么?

9.2 考虑下面的决策网络:



这个图建模了一个关于是否在两个时间作弊的决策。

假设 $P(\text{Watched}) = 0.4$, $P(\text{Trouble1} | \text{Cheat1}, \text{Watched}) = 0.8$, 对于其他的情况, Trouble1 为真的概率为 0。假设条件概率 $P(\text{Trouble2} | \text{Cheat2}, \text{Trouble1}, \text{Watched})$ 由下表给出:

<i>Cheat2</i>	<i>Trouble1</i>	<i>Watched</i>	$P(\text{Trouble2}=t)$
t	t	t	1.0
t	t	f	0.3
t	f	t	0.8
t	f	f	0.0
f	t	t	0.3
f	t	f	0.3
f	f	t	0.0
f	f	f	0.0

假设效用设置如下：

<i>Trouble2</i>	<i>Cheat2</i>	<i>Utility</i>
t	t	30
t	f	0
f	t	100
f	f	70

- (a) 变量 *Cheat2* 的最优决策函数是什么？请说明创建了哪些因子。请首先手工解答，然后使用 AIspace.org 小程序验证。
- (b) 最优策略是什么？其值为多少？列出创建的表格。
- (c) 如果被观察到 (*Watched*) 的概率增加，最优策略又是怎样的？
- (d) 如果作弊的效用减少，最优策略又是怎样的？
- (e) 当教师对于以前的作弊不是很宽恕 (不很健忘) 时，最优策略会是怎样的？

414

- 9.3 假设在一个决策网络中，决策变量 *Run* 的双亲节点为 *Look* 和 *See*。假设你使用 VE(变量消除)法来求得最优策略，且在消除了其他所有的变量后，剩下如下变量：

<i>Look</i>	<i>See</i>	<i>Run</i>	<i>Value</i>
true	true	yes	23
true	true	no	8
true	false	yes	37
true	false	no	56
false	true	yes	28
false	true	no	12
false	false	yes	18
false	false	no	22

- (a) 消除变量 *Run* 后得到的结果是什么？(提示：你不必对变量 *Run* 加和，因为它是决策变量。)
- (b) 变量 *Run* 的最优决策函数是什么？

- 9.4 假设在决策网络中有从随机变量“contaminated specimen”和“positive test”到决策变量“discard sample”的边。Sally 求解决策网络并发现其中只有一个最优策略如下：

contaminated specimen	positive test	discard sample
true	true	yes
true	false	no
false	true	yes
false	false	no

在这种情况下，你认为信息的值是怎样的？

9.5 例 9-13 所示的决策网络的结果对概率的敏感性如何？使用不同的条件概率测试这个例子并观察其对结果的影响。讨论最优策略及其期望值的灵敏性。

9.6 在例 9-13 中，假设火传感器的噪声较大，它的误肯定率有 20%：

$$P(\text{see_smoke} | \text{report} \wedge \neg \text{smoke}) = 0.2$$

且有 15% 的误否定率：

$$P(\text{see_smoke} | \text{report} \wedge \text{smoke}) = 0.85$$

此时，用它来测定烟雾还有价值吗？

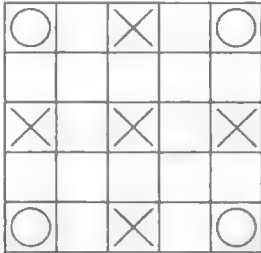
9.7 考虑习题 6-8 所示的信念网络。当观察到警报时，需要做出是否关闭反应堆的决策。关闭反应堆的代价为 c_i (该代价独立于反应堆核心是否过热)，而不关闭核心过热的反应堆会付出 c_m 的代价——这要比 c_i 高得多。

(a) 画出初始系统(即只有一个传感器的系统)中该决策问题的决策网络。

(b) 对于必须定义的新因子(在定义新因子时，表格中合适的地方应该使用 c_i 和 c_m)，请给出具体的表格。假设效用是代价的负值。

9.8 请解释在 MDP 中，为什么常常对未来的回报使用一定的折扣率。在折扣率取值分别为 0.6 和 0.9 的情况下，Agent 的行动有什么不同？

9.9 考虑下面的游戏环境：

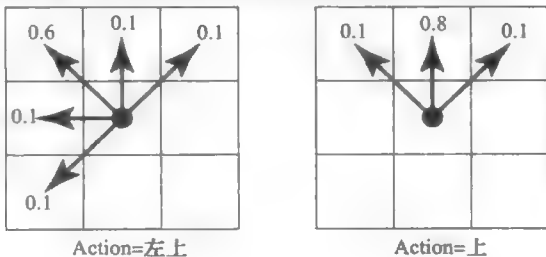


机器人可以在网格中的 25 个格子中的任一个格子。4 个角的圆圈可被视为珍宝。当机器人到达有珍宝的角落时，它能够得到 10 的回报，之后珍宝就消失了。在角落中没有珍宝时，每一步都有 $p_1 = 0.2$ 的概率会出现珍宝，且珍宝出现在每个角落的概率相同。机器人知道自己的位置，也知道珍宝的位置。

魔鬼在有叉号的格子里。在每一步，每一个魔鬼各自独立、随机地检查机器人是否进入了它的格子。如果在魔鬼做检查时，发现机器人在它的格子里，则机器人得到 -10 的回报(即丢掉 10 分)。在中间的那个格子，魔鬼在每一步检查的概率是 $p_2 = 0.4$ ；在另外 4 个标记了叉号的格子，魔鬼在每一步检查的概率为 $p_3 = 0.2$ 。

假设机器人进入一个状态后立即得到回报：也就是说，如果机器人进入一个有魔鬼的状态，它就在进入时得到(负)回报；如果机器人进入了有珍宝的状态，它就在进入该状态时得到回报，即使珍宝于同一时刻在该位置出现也是如此。

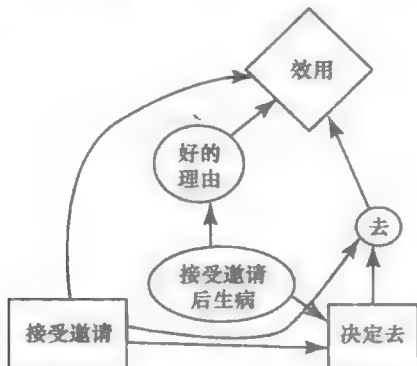
机器人有 8 个动作，对应于 8 个相邻的格子。沿斜线的动作是有噪声的动作：以 $p_4 = 0.6$ 的概率走向选定的方向，以相等的概率走向与选定方向相近的 4 个格子。竖直方向和水平方向的动作也是有噪声的动作：以 $p_5 = 0.8$ 的概率走向选定的方向，以相等的概率走向相邻的斜线方向的格子。举例来说，向左上方向和向上方向的动作的实际执行结果如下所示：



Action=左上

Action=上

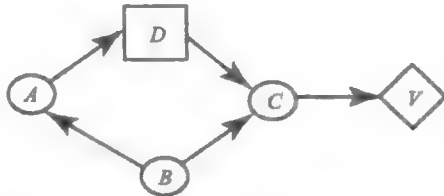
- 9.14 现实生活中，在我们不知道能否参加或者不确定是否愿意参加的情况下是否接受一个邀请是我们必须做的一个决策。下图表示了这个问题中的一个决策网络：



假设所有的决策和随机变量都是布尔值(即值域是 $\{true, false\}$)。你可以接受邀请，但当到了赴会时间，你仍然需要决定是否能赴会。你可能在接受邀请后和决定赴会前生病了。即使你决定赴会，如果你没有接受邀请则你也无法赴会。如果你生病了，那么你就有一个很好的理由不去赴会。你的效用取决于你是否接受了邀请，你是否有一个好的理由，及你是否真的赴会了。

- 给出一个表示了可能的效用函数的表格。假设唯一的最好的结果是你接受邀请、你没有一个很好的理由且你确实赴会了；唯一的最坏的结果是你接受了邀请、你没有一个很好的理由且你并没有实际赴会。给出其他的合理的效用值。
- 假设在接受邀请之前，你需要观察自己是否会生病。请注意，这个变量与你在接受邀请后生病不同。将这个情况加入网络中。你不能改变效用函数，不过新的观察必须有一个大于零的信息值。得到的网络必须是决策网络求解算法可以处理的。
- 假设在你决定是否接受原始邀请之后、实际赴会之前，你可以考查是否有一个更好的邀请(一个与原始邀请冲突的邀请，因此你无法两个都赴会)。假设你更愿意接受更好的邀请，而不是先前的邀请。(此时的决策困难是接受第一个邀请，还是等待下去直到得到更好的邀请——你可能等不到这样的邀请。)不幸的是，有另一个邀请不能成为一个好的理由。在上面的网络中，增加一个“更好的邀请”节点及相关的边以处理这个情况。(不必包含问题(b)中的节点和边。)
- 在问题(c)中，如果在“更好的邀请”和“接受邀请”之间有一条边，请解释为什么有(也就是说，环境中的哪种情况使加入这条边成为合理的?)。如果没有这条边，那么这条边应该连到哪里，同样能处理上述情景？请解释此时使得该改变合理的具体环境。
- 如果在“更好的邀请”和“接受邀请”之间没有边(不管你是否画了一个边)，那么环境中的什么因素必为真才能使缺失这条边成为合理的？

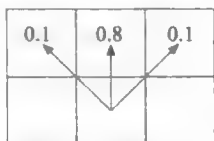
- 9.15 考虑下面的决策网络：



- 初始的因子有哪些？(不必列出对应的表格，只需要指明它们依赖于哪些变量。)
- 对于一个合法的消除排序，请说明在优化决策函数和计算期望值的过程中生成了哪些因子。在每一步，请解释消除了哪个变量、该变量是加和了还是最大化了、哪些因子组合在了一起、又生成了哪些因子(指出它们依赖的变量，不必给出取值表格)。
- 如果在决策 D 中信息 A 的值为 0，那么最优策略是什么样的？(请给出任一个最优策略的最具

体的描述。)

- 9.16 异步值迭代和标准的值迭代的最大不同是什么? 为什么异步值迭代常常比标准的值迭代工作得好?
- 9.17 考虑一个网格情景, 其中“向上”的动作有如下的动态性:



也就是说, 以 0.8 的概率向上, 分别以 0.1 的概率向左上或右上。假设有如下的状态:

s_{12}	s_{13}	s_{14}
s_{17}	s_{18}	s_{19}

进入状态 s_{14} 的回报是 +10, 进入状态 s_{19} 的回报是 -5。其他状态的回报是 0。

折扣率为 0.9。

假设我们执行的是异步值迭代算法, 存储了 $Q[S, A]$, 且对这些状态有如下的值:

$$V(s_{12}) = 5 \quad V(s_{13}) = 7 \quad V(s_{14}) = -3$$

$$V(s_{17}) = 2 \quad V(s_{18}) = 4 \quad V(s_{19}) = -6$$

在下一步异步值迭代中, 假设我们选择了状态 s_{18} 并采取向上的动作。那么 $Q[s_{18}, up]$ 的更新值是多少? 请给出算式, 不要计算或者简化它。

420

}

422

多 Agent 系统

试想一下有一个代表你个人从事电子商务的个性化软件 Agent。假设此 Agent 的任务是随时间跟踪各在线场所可销售的货物，并代表你购买其中一些有吸引力价格的货物。为了获得成功，你的 Agent 需要嵌入你对产品的偏好、你的预算和你的关于工作环境的一般知识。此外，这个 Agent 需嵌入你的关于将要进行交互的其他类似 Agent 的知识（例如，可能构成竞争拍卖的 Agent，或者代表商店业主的 Agent）——包括他们自己的偏好和知识。这些 Agent 的集合构成了一个多 Agent 系统。

——Yoav Shoham 和 Kevin Leyton-Brown[2008，第 xvii 页]

当存在拥有自己价值观并且同样推理做什么的其他 Agent 时，Agent 怎样做呢？一个智能 Agent 不应该忽略其他的 Agent，或者把它们当做环境中的噪声。我们考虑的问题是，给定一种详细描述该领域如何运作的机制来确定 Agent 应该做什么，并且设计一种拥有有用属性的机制。

10.1 多 Agent 框架

在这一章中，我们考虑有多个 Agent 的情况，其中：

- Agent 可以自主行动，每一个 Agent 都有自己的关于该领域和其他 Agent 的信息；
- 结果取决于所有 Agent 的动作，机制 (mechanism) 指定 Agent 的动作如何产生结果；
- 每个 Agent 都可以有其基于结果的自己的效用。

每个 Agent 基于自己的效用决定做什么，但它也必须和其他 Agent 进行交互。当 Agent 基于自己的目标或效用决定做什么时，称为“策略性 (strategically) 行为”。

有时候我们把自然 (nature) 当做一个 Agent。自然被定义为一个特殊的 Agent，它没有价值观，不采取策略性行为。它只是随机地动作。自然也可以看做是所有没有策略性行为的 Agent 的集合。关于 Agent 体系如图 1-3 所示，自然和其他 Agent 构成了 Agent 的环境。一个策略性的 Agent 不能把其他策略性的 Agent 当做是自然的一部分，因为它应该推理它们的效用和行为，还因为其他 Agent 有可能会与其进行合作和协商。

在多 Agent 系统研究中有两个极端：

- 完全合作的 (cooperative)，其中的 Agent 共享相同的效用函数；
- 完全竞争的 (competitive)，只有一个输，另一个才可能赢。这些通常被称为“零和博弈” (zero-sum game)，即效用可以以这样的形式表达：对于每个结果，这些 Agent 的效用的总和是零。

大多数交互是在这两个极端之间，这些 Agent 的效用可能在某些方面是互相促进的，在某些方面又是竞争的，在有些方面是相互独立的。例如，两个相邻店的商业 Agent 也许会共享拥有干净、吸引人的街区的目标；它们也会竞争客户，但是它们又对对方店的细节没有偏好。有时它们的行为互不干扰，有时它们又相互干扰。如果它们通过合作和协商来

协调它们的行动, Agent 通常可以做得更好。

在 Neumann 和 Morgenstern[1953]的开创性工作后, 大多开始使用博弈论来研究多 Agent 交互。使用博弈论, 可以研究 Agent 之间的很多交互问题。即使是相当小的博弈也可以展现很深刻的问题。然而, 博弈论的研究针对的是一般的多 Agent 交互, 而不只是人工的博弈。

多 Agent 系统在人工智能中是普遍存在的。从室内游戏, 比如跳棋、国际象棋、五子棋、围棋, 到机器人足球, 到交互式计算机游戏, 再到复杂经济系统中的 Agent, 而博弈论是人工智能不可或缺的。博弈是人工智能最早的应用之一。最早的强化学习系统之一是 Samuel[1959]的跳棋游戏, 其中第一个可操作的跳棋程序可以追溯到 1952 年。1997 年, 深蓝已经大张旗鼓地击败了国际象棋世界冠军。计算机在跳棋和五子棋上也获得了成功, 但是它在围棋上就没那么成功了, 因为过大的搜索空间和缺少优秀的可行的启发式算法。这些博弈虽然大, 但概念上是简单的, 因为 Agent 可以观察到该领域的所有状态(它们是完全可观察)。而在大多数现实世界的交互中, 领域的状态是不可观察的。

424

现在人们对部分可观察的博弈更有兴趣, 比如扑克, 它的环境是可以预测的(即使是随机的); 还有机器人足球, 它的环境不好预测。但是所有这些博弈比人们在日常生活中表现出的多 Agent 交互要简单得多, 更别说缺乏明确规则且效用包罗万象的实物交易市场了。

10.2 博弈的表示

为了实现多 Agent 交互的推理, 我们需要给出 Agent 可能行为及相应回报的表示。在经济学和人工智能中已经提出了博弈和多 Agent 交互的很多表示方案。在人工智能中, 这些表示方案通常会尝试表示博弈中可抽出用来计算收益的某些方面。

我们提出三种表示形式, 其中两种是来自经济学中的经典表示形式。第一种抽象出 Agent 的所有策略结构。第二种模型化博弈的顺序结构, 它是棋盘博弈表示的工作基础。第三种表示方案摆脱了基于状态的表示而使用基于特征的表示。

10.2.1 博弈的标准形式

博弈最基本的表示形式是博弈的策略形式(strategic form of a game)或标准形式的博弈(normal-form game)。博弈的策略形式由以下组成:

- Agent 的一个有限集合 I , I 为整数集, $I = \{1, \dots, n\}$ 。
- 每个 Agent $i \in I$ 的动作集合 A_i 。给定每个 Agent $i \in I$ 的 A_i 的赋值称为一个动作组合(action profile)。一个动作组合可以看成是一个元组 $\langle a_1, \dots, a_n \rangle$, 它表示 Agent i 实施动作 a_i 。
- u_i 为给定一个动作组合时, 每个 Agent $i \in I$ 的效用函数, 表示对于一个给定动作组合的 Agent i 返回期望效用。

所有 Agent(一个动作组合)的共同作用产生出一个结果。每个 Agent 在每个结果上都有一个效用。Agent 的效用是指 Agent 所感兴趣的一切, 包括公平和社会福祉。因此, 我们假设每个 Agent 试图使自己的效用最大化, 而不关心其他 Agent 的效用。

【例 10-1】石头剪子布是一种常见的儿童游戏, 甚至还有一个石头剪子布的世界冠军。假设有两个 Agent(玩家), Alice 和 Bob。每个 Agent 有三个动作, 所以

425

$$A_{\text{Alice}} = A_{\text{Bob}} = \{\text{rock}, \text{paper}, \text{scissors}\}.$$

对 Alice 和 Bob 每个动作组合，Alice 会有一个效用，Bob 也会有一个效用。通常我们用图 10-1 所示的表来表示，这就是所谓的收益矩阵 (payoff matrix)。Alice 选择行，Bob 选择列。这给出了一个数字对：第一个数字是行上的玩家 (Alice) 的收益，第二个是列上的玩家 (Bob) 的收益。需要注意的是，它们中的每个玩家的效用取决于双方玩家的动作。动作组合的一个例子是 $\langle \text{scissors}_{\text{Alice}}, \text{rock}_{\text{Bob}} \rangle$ ，即 Alice 选择出剪子，Bob 选择出石头。在这个动作组合中，Alice 得到的效用为 -1，Bob 得到的效用为 1。这是个零和博弈，因为只有一方输了另一方才能赢。

		Bob		
		石头	布	剪刀
Alice	石头	0, 0	-1, 1	1, -1
	布	1, -1	0, 0	-1, 1
	剪刀	-1, 1	1, -1	0, 0

图 10-1 石头剪子布游戏的策略形式

博弈的这个表示形式有很大的局限性，因为它只基于每个 Agent 的单一动作给出了一次性收益，这些动作同时被每一个 Agent 所选择。然而，动作定义的解释却很宽泛。

通常情况下，“动作”不只是一个简单的选择，而是一种策略：在各种突发情况下，Agent 将要做什么的一种规范。从本质上讲，其标准形式是给定 Agent 可能策略的效用的规范。这就是它被称为博弈的策略形式的原因。

在一般情况下，“动作”在标准形式博弈中的定义是 Agent 的控制器。因此，每个 Agent 选择一个控制器，在环境中运行的每个 Agent 控制器产生预期的结果就是效用。虽然后面的例子中是简单的动作，但是一般情况下每个 Agent 可能采取的动作 (可能的控制器) 的数量是巨大的。

10.2.2 博弈的扩展形式

鉴于博弈的标准形式把控制器表示为单个单位，那么在时间上展开博弈的规范更自然。博弈的扩展形式是单个 Agent 决策树的扩展。我们首先给出一个定义，假定这个博弈是完全可观察的 (称为博弈论中的完全信息)。

完全信息博弈的扩展形式 (extensive form) 或博弈树 (game tree) 是一个有限树，其中树中的节点是状态，弧对应的是 Agent 的动作。实际上：

- 每个内部节点被标记为 Agent (或者 nature)。Agent 用来控制节点。
- 从标记为 Agent i 的节点出来的每个弧对应的是 Agent i 的行为。
- 每个标记为 nature 的内部节点都有关于它的子节点的概率分布。
- 叶子节点表示最后结果并用 Agent 的效用来标记。

一个博弈的扩展形式指定了一个特定的博弈展开。每个到叶子节点的路径称为一次运行 (run)，指定了通过依赖于 Agent 或 nature 的选择使博弈能够进行下去的一条特定路径。

Agent i 的一个纯策略是一个从被 Agent i 控制的节点到动作的函数。也就是说，一个纯策略为每个 Agent i 控制的节点选择子节点。一个策略组合 (strategy profile) 包含了每个 Agent 的策略。

【例 10-2】试想一个共享的博弈，有两个 Agent，分别为 Andy 和 Barb，并且有两个相同的東西要分给他俩。Andy 首先选择如何分配它们：Andy 都保留，或者选择分享使每

人均得到一个, 或者都给 Barb。然后 Barb 可以拒绝这次分配从而他们两个什么也得不到, 或者他接受分配从而他们两个都得到了相应数量。

图 10-2 中表示出了共享博弈的扩展形式。Andy 有 3 个策略。Barb 有 8 个纯策略, 每个策略是它所控制节点的分配的组。结果共有 24 个策略组合。

给定一个策略组合, 每个节点对每个 Agent 都有一个效用。一个节点上的 Agent 的效用从下往上递归定义:

- 叶子节点上每个 Agent 的效用给定为叶子节点的一部分。
- 被 Agent i 控制的某个节点上 Agent j 的效用是由 Agent i 的策略所选择的子节点上的 Agent j 的效用。
- 被 nature 控制的节点上的 Agent i 的效用是其子节点上的 Agent i 效用的期望值。也就是说, $u_i(n) = \sum P(c)u_i(c)$, 其中的求和包括节点 n 上所有的子节点 c , $P(c)$ 是 nature 选择子节点 c 的概率。

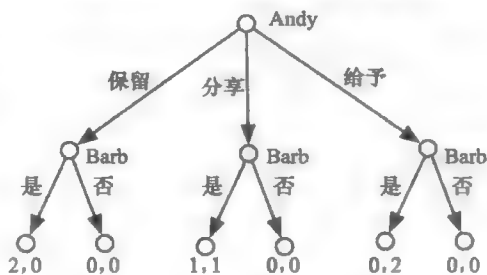


图 10-2 共享博弈的扩展形式

427

【例 10-3】在共享博弈中, 假设我们有以下策略组合: Andy 选择 *keep*, 且 Barb 对她会选择的每个节点选择 *no*, *yes*, *yes*。根据这个策略组合, 在最左内部节点上 Andy 的效用是 0, 在中间内部节点上 Andy 的效用是 1, 在最右内部节点上 Andy 的效用是 0。在根节点上 Andy 的效用是 0。

前面定义的博弈的扩展形式假定 Agent 可以观察到世界状态(也就是说, 它们知道每一步是什么节点)。这意味着, 博弈的状态必须是充分可观察的。在一个部分可观察博弈或者一个不完全信息博弈中, Agent 并不一定知道博弈的状态。为了建模这些博弈的扩展形式, 我们介绍下信息集的概念。一个信息集(information set)是一个节点集合, 它们都被同一个 Agent 控制并且拥有同一个可用的动作集合。这意味着 Agent 无法区分信息集中的元素。Agent 只知道这个博弈的状态是在信息集中的某个节点上, 但不知道具体在哪个节点上。在一个策略中, Agent 为每个信息集选择一个动作; 在信息集中的每个节点上进行同样的行为。因此, 在扩展形式中, 一个策略指定了从信息集到动作的一个函数。

【例 10-4】图 10-3 给出了例 10-1 的石头剪刀布游戏的扩展形式。信息集的元素在一个圆角矩形中。Bob 必须为信息集中的每个节点选择相同的动作。

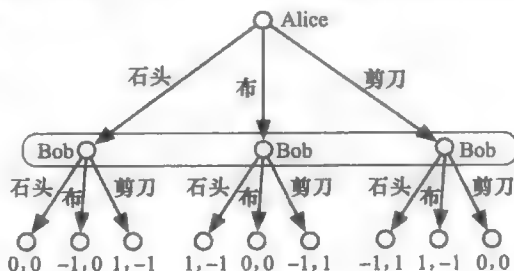


图 10-3 “石头剪刀布”博弈的扩展形式

10.2.3 多 Agent 决策网络

博弈的扩展形式可以看做博弈的基于状态的表示形式。正如我们在前面已经看到的, 在特性方面来描述状态往往更简洁。多 Agent 决策网络(multiagent decision network)是多 Agent 决策问题的一个因素表示形式。除了每个决策点被标记为 Agent, 其为节点选择一个值, 这类似于决策网络。每个 Agent 有一个效用节点, 它指定了这个 Agent 的效用。当

428

Agent 必须采取行动时，决策节点的父节点指定该 Agent 可用的信息。

【例 10-5】 图 10-4 给出了一个消防部门的多 Agent 决策网络的例子。在此方案中，有两个 Agent，Agent 1 和 Agent 2。每个都拥有辨别是否发生火灾的噪声传感器。然而，如果它们都呼叫，它们的呼叫可能会互相干扰导致每个呼叫都不管用。Agent 1 可以选择决策变量 $Call1$ 的值，并且只能观察到变量 $Alarm1$ 的值。Agent 2 可以选择决策变量 $Call2$ 的值，并且只能观察到变量 $Alarm2$ 的值。该呼叫是否工作取决于 $Call1$ 和 $Call2$ 的值，而消防部门是否到来取决于呼叫是否工作。Agent 1 的效用取决于三个方面：是否有火灾、消防部门是否到来和它们是否呼叫——Agent 2 也是如此。

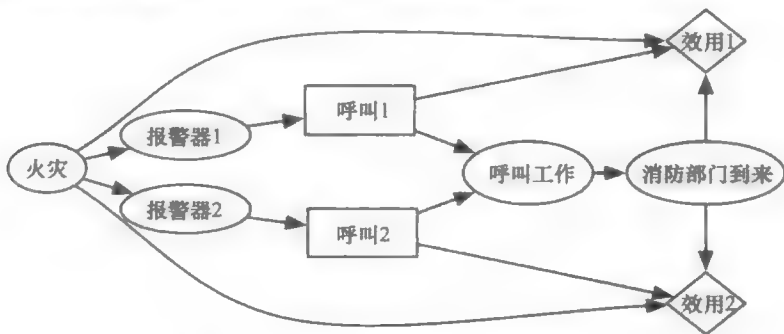


图 10-4 火灾事例的多 Agent 决策网络

一个多 Agent 决策网络能被转换成一个标准形式的博弈；然而，策略的数量可能是巨大的。如果一个决策变量有 d 个状态， n 个二进制的父节点，那么父节点有 2^n 个分配值，有 d^{2^n} 个策略。这仅仅是一个单一决策节点，更多复杂网络在转换为标准形式时会更加庞大。因此，我们提出的与策略数量成指数关系的算法不可能对任意情况可行，只适用于最小的多 Agent 决策网络。

还有利用其他结构来表示多 Agent 系统的形式。例如，一个 Agent 的效用也许会取决于采取某种动作的 Agent 的数量，而不是取决于它们自身。一个 Agent 的效用也许会依赖于一小部分 Agent 的行为，而非所有其他 Agent 的行为。一个 Agent 的效用也许仅仅取决于临近位置 Agent 做什么，而不是这些 Agent 的本体或者其他 Agent 做什么。

429

10.3 完全信息的计算策略

多 Agent 的完全可观察性相当于拥有**完全信息**(perfect information)。在完全信息博弈中，Agent 按顺序行动，当一个 Agent 要行动时，它会在决定做什么之前观察这个世界的状态。每个 Agent 均以使自己的效用最大化为目的采取行动。

完全信息博弈可以扩展形式的博弈来表示，其中信息集只包含一个节点。它们也可以表采用一个多 Agent 决策网络表示，其中决策节点是完全排序的，且对每个决策节点，其父节点包含前继决策节点和它们所有的父节点(即它们是没有遗忘的决策网络)。

完全信息博弈也可以用类似完全可观察的单一 Agent 系统的方法来解决。我们可以用向后的动态规划或者向前的搜索来解决。与单一 Agent 的情况不同的是，多 Agent 算法为每个 Agent 每次的移动保留一个效用，它选择一个动作使 Agent 移动效用最大化。动态规划的变体——称为**逆向归纳法**(backward induction)——本质上遵循着 Agent 节点的效用定义，但是在每个节点上，控制节点的 Agent 要选择使得它的效用最大化的动作。

【例 10-6】考虑图 10-2 的共享博弈。对每个标记为 Barb 的节点，她要选择使她效用最大化的值。因此，她会为她所控制的右面两个节点选择“yes”，要么她就将要选择她控制的最左面的节点了。假设她对这个节点选择“no”，那么 Andy 就要选择下列行为中的一个：keep 对他的效用是 0，share 的效用是 1，give 的效用是 0，所以他选择共享。◀

在两个 Agent 的竞争环境中，一个 Agent 有正的收益，另一个 Agent 则为负的收益，这是一个 2-Agent 的零和博弈。这种博弈的特征可以用一个数字来表述，一个 Agent 试图使该值最大化，而另一个 Agent 试图使其最小化。对于一个 2-Agent 零和博弈，该单一值会引出一个极大极小(minimax)策略。如果是被试图最大化的 Agent 控制的节点，那么每个节点是一个 MAX 节点；或者如果是被试图最小化的 Agent 控制的节点，那么每个节点是一个 MIN 节点。

逆向归纳法可用来找到最佳的极大极小策略。从下往上，逆向归纳法在 MAX 节点最大化，在 MIN 节点最小化。然而，逆向归纳法需要对整个博弈树进行遍历。通过剪枝掉不可能成为最优方案的那部分搜索树来实现。

【例 10-7】考虑图 10-5 中的博弈树搜索。在该图中，方形 MAX 节点由最大化 Agent 控制，圆形 MIN 节点由最小化 Agent 控制。

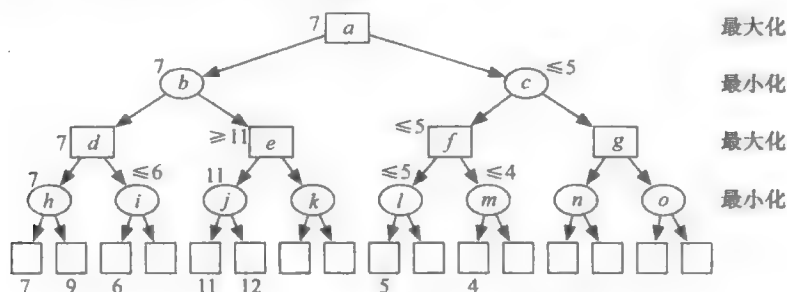


图 10-5 零和博弈树的节点剪枝

假设给定博弈的定义的情况下，叶子节点的值可以给定或者可以计算出来。正如图中所示，底部的数字显示了部分值，其他值忽略。假设我们对这个树进行从左开始深度优先遍历。节点 h 的值是 7，因为它是 7 和 9 中的最小值。仅仅通过考虑 i 的最左子节点，其值是 6，我们知道 i 的值是小于或等于 6 的。因此，在节点 d ，最大化 Agent 将向左走。我们没有必要评估 i 的其他子节点。相似的， j 的值是 11，所以 e 的值至少是 11，在节点 b 的最小化 Agent 将会选择向左走。

l 的值小于或等于 5， m 的值小于或等于 4；因此， f 的值小于或等于 5，所以 c 的值将会小于或等于 5。所以，在节点 a ，最大化 Agent 将会选择向左走。要注意的是，这种说法不取决于没有数字的叶子节点的值，它也不取决于没有探讨的子树的大小。◀

前面的例子分析了什么样的节点可以被剪枝。极大极小 α - β 剪枝是一个深度优先搜索算法，它通过向下传递参数 α 和 β 的剪枝信息来剪枝。在这个深度优先搜索中，节点有“当前”值，这个值由它的子孙节点取得。当它得到更多有关它其他子孙节点值的信息时，这个当前值可以随之更新。

参数 α 可以用于剪枝 MIN 节点。最初，对于当前节点的所有 MAX 祖先，它是最大的当前值。任何当前值小于等于它的 α 值的 MIN 节点没有必要更进一步探讨。在前面的例子里，采用这个思路来剪枝 l 、 m 和 c 的其他后代节点。

与之相对的是参数 β ，它可以用来剪枝 MAX 节点。

图 10-6 给出了 $\alpha\beta$ 剪枝的极大极小算法。它起初被称为

MinimaxAlphaBeta($R, -\infty, \infty$)

其中 R 是根节点。要注意的是, 它使用 α 作为 MAX 节点的当前值, β 作为 MIN 节点的当前值。

【例 10-8】 考虑在图 10-5 中的树上运行 *MinimaxAlphaBeta*。我们将展示递归调用。最初, 它调用

MinimaxAlphaBeta($a, -\infty, \infty$)

然后依次调用

MinimaxAlphaBeta($b, -\infty, \infty$)

MinimaxAlphaBeta($d, -\infty, \infty$)

MinimaxAlphaBeta($h, -\infty, \infty$)

最后一次调用查找它的两个子节点中最小的那个, 并且返回值 7。接下来程序调用

MinimaxAlphaBeta($i, 7, \infty$)

然后得到 i 的第一个子节点的值是 6。

由于 $\alpha \geq \beta$, 返回值 6。然后 d 的调用返回值 7, 并且它调用

MinimaxAlphaBeta($e, -\infty, 7$)

节点 e 的第一个子节点返回值 11, 并且由于 $\alpha \geq \beta$, 返回值 11。然后 b 返回值 7, 并且 a 调用了

MinimaxAlphaBeta($c, 7, \infty$)

接着调用

MinimaxAlphaBeta($f, 7, \infty$)

最终返回值 5, 所以 c 的调用返回值 5, 所以最终整个程序返回值 7。

通过跟踪值, 最大化 Agent 知道在 a 处向左走, 然后最小化 Agent 在 b 处将向左走, 等等。

该算法提供的剪枝量取决于每个节点的子节点的排序。如果一个 MAX 节点的最高值子节点首先被选中, 并且一个 MIN 节点的最低值首先被返回, 那它的效果最好。在真正博弈的实现中, 很多的努力都是要保证这一结果。

大多数真实的博弈即使使用 $\alpha\beta$ 剪枝也无法进行极大极小搜索。对于这些博弈, 通常可在任何节点停止, 而不再是叶子节点。在算法停止的节点上返回的值即是对这个节点的一个评估值。用来评估这个值的函数叫**评价函数**(evaluation function)。已有大量寻找优秀的评价函数的工作。此类工作有一点需要注意, 在计算评价函数所需要的计算量与在给定的时间内可搜索的搜索空间的大小上需有一个权衡。至于如何在复杂的计算和巨大搜索空间之间进行权衡就是个经验问题了。

10.4 部分可观察的多 Agent 推理

部分可观察性是指一个 Agent 不知道世界的完全状态或者所有的 Agent 同时行动。

部分可观察下的多 Agent 的情况比完全可观察下的多 Agent 的情况或部分可观察下的单一 Agent 的情况更复杂。下面这个简单的例子显示了即使仅有两个 Agent 并且每个 Agent 只有很少选择的情况下也会出现的一些重要问题。

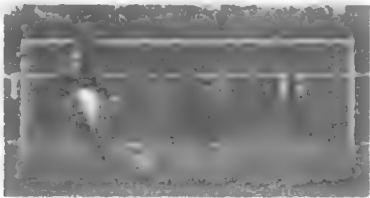
```

1: procedure MinimaxAlphaBeta( $N, \alpha, \beta$ )
2:   Inputs
3:      $N$ : 博弈树中的一个节点
4:      $\alpha, \beta$ : 实数
5:   Output
6:     节点  $N$  的值
7:   if  $N$  是叶子节点 then
8:     return  $N$  的值
9:   else if  $N$  是 MAX 节点 then
10:    for each  $N$  的子节点  $C$  do
11:      Set  $\alpha \leftarrow \max(\alpha, \text{MinimaxAlphaBeta}(C, \alpha, \beta))$ 
12:      if  $\alpha \geq \beta$  then
13:        return  $\beta$ 
14:    return  $\alpha$ 
15:   else
16:    for each  $N$  的子节点  $C$  do
17:      Set  $\beta \leftarrow \min(\beta, \text{MinimaxAlphaBeta}(C, \alpha, \beta))$ 
18:      if  $\alpha \geq \beta$  then
19:        return  $\alpha$ 
20:    return  $\beta$ 

```

图 10-6 $\alpha\beta$ 剪枝中的极大极小算法

【例 10-9】 考虑如图 10-7 所描述情况，在足球比赛中罚点球。如果射门者向他的右边踢并且守门员向他的右边跳起，进球的概率是 0.9，图中同样给出了其他的动作组合。



		守门员	
射门者	左	左	右
	右	0.6	0.2
		0.3	0.9

进球的概率

图 10-7 点球大战。射门者可选择向左或者向右射，守门员可以选择向左或者向右扑

当射门者希望进球概率最大而守门员希望进球概率最小的情况下，射门者应该怎么做？射门者可能会认为最好往他的右边踢，因为往右踢的那对数字比往左踢的那对数字要大。守门员可能会认为如果射门者往右踢，那他应该往左跳。然而，如果射门者认为守门员要想左跳，那他应该往左踢。但是随后，守门员应该往右跳，然后射门者应该往右踢。

433

每个 Agent 可能会面对推理另一个 Agent 将要做什么的无限回归。在它们推理的每个阶段中，Agent 会反转它们的决策。然而，人们可以想象在某一处中断，那么该动作就单纯地是一个任意深度的函数。更糟的是，如果射门者知道守门员推理的深度限制，他就可以利用这个知识来决定射门者要做什么并且适当地选择他的动作。

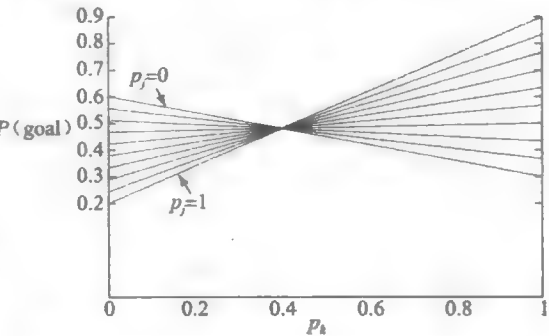
Agent 的另一个方法是随机地选择动作。你可以想象射门者和守门员各自偷偷地扔硬币来决定做什么。然后你应该考虑硬币是否是有偏向的。假设射门者决定以概率 p_k 往他的右面踢，守门员决定以概率 p_j 往他的右面跳。那么进球的概率是

$$0.9p_kp_j + 0.3p_k(1 - p_j) + 0.2(1 - p_k)p_j + 0.6(1 - p_k)(1 - p_j)$$

图 10-8 将进球的概率表示为 p_k 的函数。不同的行对应于 p_j 的不同值。

当值 $p_k = 0.4$ 时有些特殊。在这个值中，进球的概率是 0.48，它独立于 p_j 的值。也就是说，不管守门员做什么，射门者进球的预期概率是 0.48。如果射门者偏离了 $p_k = 0.4$ ，他可以做得更好或者更坏，但这取决于守门员做什么。

对于与 $p_j = 0.3$ 相交的所有行， p_j 的情况也是类似的。再次，当 $p_j = 0.3$ 时，进球的概率是 0.48。



$p_k = 0.4$ 且 $p_j = 0.3$ 的策略是特殊的，

图 10-8 将进球概率表示为动作概率的函数

在这个意义上，Agent 通过单方面偏离这个策略都不能做得更好。然而，这并不意味着他们不能做得更好；如果 Agent 中的一个偏离了这种均衡，另一个 Agent 通过偏离这种均衡可以做得更好。但是，这种均衡对于 Agent 是安全的，意义在于即使另外一个 Agent 知道一个 Agent 的策略，另外一个 Agent 也不能强制这个 Agent 得到更坏的结果。使用这个策略意味着一个 Agent 不必担心对另一个 Agent 双向猜测。他将会得到他所能保证得到的最好回报。

434

现在让我们扩大策略的定义使其包括随机策略。

考虑博弈的标准形式，其中每个 Agent 要同时选择一个动作。每个 Agent 在不知道另一个 Agent 选择的情况下选择一个动作。

一个 Agent 的策略是该 Agent 动作的一个概率分布。如果这个 Agent 行动非常确定，其中一个概率是 1，剩下的是 0，这叫做一个**纯策略**(pure strategy)。如果这个 Agent 没有纯策略，其中没有一个概率是 1，并且将有一个以上动作的概率是非零的，这叫做**随机策略**(stochastic strategy)。在一个策略中非零概率的动作集叫做这个策略的**支持集**(support set)。

一个**策略组合**(strategy profile)是对每个 Agent 的一个策略分配。如果 σ 是一个策略组合， σ_i 是在 σ 中 Agent i 的策略， σ_{-i} 是其他 Agent 的策略。那么 σ 是 σ_i, σ_{-i} 。如果这个策略组合由纯策略组成，它通常叫做一个**动作组合**(action profile)，因为每个 Agent 都有一个特定的动作。

435

一个策略组合 σ 对每个 Agent 有一个效用。 $utility(\sigma, i)$ 是策略组合 σ 对 Agent i 的效用。当给定构成组合的基本动作的效用以及动作的概率时，通过计算期望效用可以得出一个随机策略组合的效用。

Agent i 对其他 Agent 的策略 σ_{-i} 的一个**最好回应**(best response)是对该 Agent 有着最大效用的策略。也就是说， σ_i 是 σ_{-i} 的最好回应，当对 Agent i 所有其他的策略 σ'_i 有

$$utility(\sigma_i, \sigma_{-i}, i) \geq utility(\sigma'_i, \sigma_{-i}, i)$$

对每个 Agent i 来说，如果策略 σ_i 是 σ_{-i} 的最好回应，那么这个策略组合 σ 就是一个**纳什均衡**(Nash equilibrium)。也就是说，一个纳什均衡就是一个策略组合，任何 Agent 通过单方面偏离这个组合不可能做得更好。

Nash[1950]的一个伟大成果是每个有限的博弈至少拥有一个纳什均衡点。

【例 10-10】 在例 10-9 中，当 $p_k=0.4$ ， $p_j=0.3$ 时有唯一的纳什均衡。它的含义是如果射门者的 $p_k=0.4$ ，那无论守门员做什么无所谓；守门员都会有同样的回报，所以 $p_j=0.3$ 是一个最好回应(任何其他策略也一样)。同样，如果守门员的 $p_j=0.3$ ，那无论射门者做什么无所谓；每一个策略也一样，包括 $p_k=0.4$ ，是一个最好回应。Agent 可以在两个动作中随机选择唯一的原因是动作是否有同样的期望效用。两个动作的所有的随机混合具有同样的效用。为混合概率选择一个特定值的原因是为了防止其他 Agent 利用这种偏离。

有很多多个纳什均衡点的例子。考虑下面的 2-Agent 和 2-动作的博弈。

【例 10-11】 假设两个 Agent 要竞争一个资源。每个 Agent 可以选择像一只鹰或者一只鸽子来行动。假设这个资源价值为 R 个单元， $R>0$ 。如果两个 Agent 都像鸽子动作，则它们共享这个资源。如果一个 Agent 像鹰动作另一个像鸽子动作，那鹰 Agent 会得到资源而鸽子 Agent 什么也得不到。如果它们都像鹰动作，就会破坏资源并且每个 Agent 将得到回报为 $-D$ ，其中 $D>0$ 。这可以由下面的收益矩阵来描述：

		Agent 2	
		鸽子	鹰
Agent 1	鸽子	$R/2, R/2$	$0, R$
	鹰	$R, 0$	$-D, -D$

在这个矩阵中，Agent 1 选择行，Agent 2 选择列，在单元格里的收益由 Agent 1 和 Agent 2 的回报对组成。每个 Agent 要试着最大化它自身的回报。

在这个博弈中有三个纳什均衡：

- 第一个均衡中，Agent 1 像鹰动作，Agent 2 像鸽子动作。Agent 1 不想要偏离，因为那样的话它们就必须共享资源。Agent 2 不想要偏离，因为那样的话就有破坏。
- 第二个均衡中，Agent 1 像鸽子那样动作，Agent 2 像鹰那样动作。

- 第三个均衡中，两个 Agent 都随机地动作。在这个均衡中，有一些破坏的几率。像鹰那样动作的概率随着资源的 R 值上升，随着破坏的 D 值增加而下降。参看习题 10.1。

在这个例子中，你可以想象每个 Agent 装腔作势地要表示它将会做什么，以此试图迫使均衡有利于它。

拥有多个纳什均衡并非来自对手的情况，如下图的示例所示。

【例 10-12】 假设有两个想在一起的人。Agent 1 更偏好一起去看足球比赛，Agent 2 更偏好一起去购物。如果他们不在一起的话他俩都不高兴。假设他俩必须要同时选择做什么。用下面的收益矩阵来表示：

		Agent 2	
		足球	购物
Agent 1	足球	2, 1	0, 0
	购物	0, 0	1, 2

在这个矩阵中，Agent 1 选择行，Agent 2 选择列。

在这个博弈中，有三个纳什均衡。一个均衡是他们都去购物，一个是他们都去看足球比赛，一个是随机策略。

这是一个协调(coordination)的问题。知道了均衡集并不能确定哪个 Agent 要做什么，因为一个 Agent 要做什么取决于另一个 Agent 要做什么。在这个例子中，你可以想象他们决定选择哪个均衡的谈话。

即使存在一个唯一的纳什均衡，纳什均衡也不能保证给每个 Agent 的最大收益。下面的例子是囚徒困境(prisoner's dilemma)的另一种形式。

【例 10-13】 试想一下，你和一个你将再也不会遇到的陌生人共同参加一个电视游戏，你们每个人都有选择

- 自己拿 100 美元。
- 给另一个人 1 000 美元。

这可以用下面的收益矩阵来描述：

		Player 2	
		拿	给
Player 1	拿	100, 100	1 100, 0
	给	0, 1 100	1 000, 1 000

437

无论对方 Agent 如何选择，每个 Agent 自身选择拿钱均比给钱更好。然而，两个 Agent 都给钱却比都拿钱更好。

因此，当两个 Agent 都选择拿钱时，有一个唯一的纳什均衡。这个策略组合的结果是每个参赛者拿到 100 美元。两个参赛者都给钱的策略组合结果是每个参赛者拿到 1 000 美元。然而，在这个策略组合中，每个 Agent 会因为偏离而获得额外回报($1\ 100 - 1\ 000$)。

关于囚徒困境有很多研究，因为看上去贪婪不太合理，其中的每个 Agent 为了自己的最大利益去做，结果导致每人更糟糕。当博弈进行多次后，Agent 会更偏好给钱，这就是所谓的连续囚徒博弈(sequential prisoner's dilemma)。连续囚徒博弈的一个策略是针锋相对(tit-for-tat)策略：每个参与者一开始采取给钱，然后在下面的每一步采取另一个 Agent 先前的行为。只要两个参与者都不知道最后的动作，这个策略就是一个纳什均衡(见习题 10.3)。

不仅仅在部分可观察的博弈中有多个纳什均衡，甚至在一个完全信息博弈中也可能有多个纳什均衡，有时甚至是无数个纳什均衡，如下面的示例所示。

【例 10-14】 考虑例 10-2 的共享博弈。在这个博弈中有无数个纳什均衡。有一组均衡，其中 Andy 共享，Barb 在中心节点对于共享选择“是”，并且可以随机地做其他选择，只要对向左选择说“是”的概率小于或等于 0.5。在这些纳什均衡中，他们都能得到 1。有另一组纳什均衡，Andy 保留，Barb 随机地做出选择，以便在左分支说“是”的概率大于或等于 0.5。在这些均衡中，Barb 得到 0，Andy 得到在 $[1, 2]$ 中的某个值，这取决于 Barb 的概率。第三组纳什均衡中，Barb 在最左节点选择“是”的概率是 0.5，在中心节点选择“是”，而 Andy 以任何概率在“keep”和“share”之间随机选择。

假设稍微修改一下这个共享博弈，让 Andy 贿赂 Barb 让他说“是”。这个可以通过将收益 2, 0 变为 1.9, 0.1 做到。Andy 可能会认为：“如果在 0.1 和 0 之间选择，Barb 会选择 0.1，所以我应该保留。”但是 Barb 可能想：“我应该对 0.1 说不，这样 Andy 会共享我能得到 1。”在这个例子中（甚至忽略最右边的分支）有多个纯纳什均衡，其中一个是 Andy 保留并且 Barb 在最左分支说是。在这个均衡中，Andy 得到 1.9，Barb 得到 0.1。还有另外一个纳什均衡，Barb 在最左选择节点处说不，并且在中心分支处说是，Andy 选择共享。在这个均衡中，他们都得到 1。这似乎是对 Barb 有利。但是，Andy 可能认为 Barb 在做空的威胁。实际上如果他选择保留，Barb 为了最大化她的效用不会说“不”。

在修改后的共享博弈中逆向归纳算法只能找到一个均衡。它计算出了一个子博弈完美均衡，它假设 Agent 在每一个要选择的节点处选择对它们有最大的效用的动作。它假设 Agent 在不符合它们的利益的时候不进行威胁。在前面修改后的共享博弈的例子中，它假设 Barb 对那个小贿赂说“是”。然而，在与真正的对手交锋的时候，我们必须要注意的是，他们是否会跟进我们可能认为不合理的威胁。

10.4.1 纳什均衡计算

想要为一个策略形式的博弈计算纳什均衡，需以下三个步骤：

- 1) 剔除劣势策略；
- 2) 确定哪些动作将拥有非零概率，这就是所谓的**支持集**；
- 3) 确定支持集中动作的概率。

事实证明，第二步是最难的。

1. 剔除劣势策略

对于其他 Agent 的每一个行为，如果 Agent A 的 s_1 的效用比 Agent A 的 s_2 的效用高，则 Agent A 的策略 s_1 就占优于策略 s_2 。被另一个策略占优的任何纯策略都可以剔除，不予考虑。占优的策略可以是一个随机的策略。可以重复进行剔除。

【例 10-15】 考虑下面的收益矩阵，其中第一个 Agent 选择行，第二个 Agent 选择列。在每个单元格里是一个收益对：Agent 1 的收益和 Agent 2 的收益。Agent 1 有动作 $\{a_1, b_1, c_1\}$ 。Agent 2 有动作 $\{d_2, e_2, f_2\}$ 。

		Agent 2		
		d_2	e_2	f_2
Agent 1	a_1	3, 5	5, 1	1, 2
	b_1	1, 1	2, 9	6, 4
	c_1	2, 6	4, 7	0, 8

(在看解决方案之前试着弄清每个 Agent 应该做什么。)

动作 c_1 可以删除, 因为它被动作 a_1 占优: 如果动作 a_1 可以做, Agent 1 永远不会做 c_1 。你可以看到无论另一个 Agent 做什么, Agent 1 做 a_1 的收益比做 c_1 的收益要大。

一旦动作 c_1 被剔除, 动作 f_2 也可以被剔除, 因为它被随机策略 $0.5 \times d_2 + 0.5 \times e_2$ 占优。

439

一旦 c_1 和 f_2 被剔除掉, b_1 被 a_1 占优, 所以 Agent 1 将会做动作 a_1 。由于 Agent 1 要做 a_1 , Agent 2 要做 d_2 。因此, 这个博弈中 Agent 1 的收益将是 3, Agent 2 的收益将是 5。◀

如果对于其他 Agent 的所有动作组合 σ_{-i} ,

$$utility(s_1\sigma_{-i}, i) > utility(s_2\sigma_{-i}, i)$$

那么 Agent i 的策略 s_1 严格占优 (strictly dominate) 于策略 s_2 。显而易见的是, 即使 s_1 是一个随机策略, 如果 s_2 是被某个策略 s_1 严格占优的纯策略, 那么 s_2 永远不会出现在任何纳什均衡的支持集中。重复剔除被严格占优的策略具有相同的结果, 在操作中不必考虑剔除的次序。

也有“弱占优”的概念, 即在上述公式中的大于号被替换成大于或等于。如果利用弱占优概念, 则总存在一个由非占优策略支持的纳什均衡。然而, 一些纳什均衡可能会丢失。此外, 哪些均衡会丢失取决于剔除劣势策略的顺序。

2. 计算随机策略

我们可以利用这样一个事实, 如果所有的动作对 Agent 有相同的效用 (给定另一个 Agent 的策略), 则该 Agent 只能在动作中随机选择。这形成了一组约束, 通过解约束来得到一个纳什均衡。如果这些约束可以用 $(0, 1)$ 范围中的数字来解决, 并且为每个 Agent 所计算的混合策略不会被 Agent 的另一个策略所占优, 那么这个策略组合是一个纳什均衡。

回顾一个支持集是这样一個纯策略集, 即在纳什均衡中每个策略都有非零的概率。

一旦劣势策略被剔除, 我们可以搜索支持集来决定支持集是否将形成一个纳什均衡。注意, 如果一个 Agent 有 n 个可采取的动作, 那么就有 $2^n - 1$ 个非空子集, 并且我们必须搜索不同 Agent 的支持集的组合。因此, 除非没有非劣势动作或者有较小支持集的纳什均衡, 这是不可行的。为了找到简单的 (在支持集中动作的数量方面) 均衡, 我们可以从小到大搜索支持集。

假设 Agent i 在一个纳什均衡中的动作 $a_i^1, \dots, a_i^{k_i}$ 中随机选择。 p_i^j 是 Agent i 做动作 a_i^j 的概率。 $\sigma_{-i}(p_{-i})$ 是其他 Agent 的概率函数形式的策略。事实上, 一个纳什均衡具有以

下约束: $p_i^j > 0$, $\sum_{j=1}^{k_i} p_i^j = 1$, 并且对于所有的 j, j'

$$utility(a_i^j\sigma_{-i}(p_{-i}), i) = utility(a_i^{j'}\sigma_{-i}(p_{-i}), i)$$

440

我们还可以要求做动作 a_i^j 的效用不小于支持集外的动作的效用。因此, 对于所有的 $a_i^{j'} \notin \{a_i^1, \dots, a_i^{k_i}\}$,

$$utility(a_i^j\sigma_{-i}(p_{-i}), i) \geq utility(a_i^{j'}\sigma_{-i}(p_{-i}), i)$$

【例 10-16】 在例 10-9 中, 假设守门员以 p_j 的概率向右跳并且射门者以 p_k 的概率向右踢。

如果守门员向右跳, 进球的概率是

$$0.9p_k + 0.2(1 - p_k)$$

如果守门员向左跳, 进球的概率是

$$0.3p_k + 0.6(1 - p_k)$$

唯一的一次守门员可以随机选择的情况是以上两者相等, 即如果

$$0.9p_k + 0.2(1 - p_k) = 0.3p_k + 0.6(1 - p_k)$$

求解 p_k 得 $p_k=0.4$ 。

相似的，如果射门者随机选择，无论射门者往左踢还是往右踢，进球的概率肯定是一样的：

$$0.2p_j + 0.6(1-p_j) = 0.9p_j + 0.3(1-p_j)$$

求解 p_j 得 $p_j=0.3$ 。

因此，唯一的纳什均衡是 $p_k=0.4$ ， $p_j=0.3$ 。

10.4.2 学习协调

由于存在多个均衡，在许多情况下，即使它知道这个博弈的所有后果和 Agent 的效用，一个 Agent 实际上并不明确应该做什么。然而，大多数实际的策略相遇更难，因为 Agent 不知道后果以及其他 Agent 的效用。

另一种由计算纳什均衡所隐含的深层次策略性推理的方法是尝试学习执行什么样的动作。这和第 7 章的学习的标准设定是完全不同的，标准设定中 Agent 了解学习的是一些未知但是固定的概念；在这里，一个 Agent 学习与其他正在学习中的 Agent 交互。

本节给出了一个简单的算法，可以用来逐步提高一个 Agent 的策略。我们假设，Agent 反复地玩同一个博弈并且根据它们做得怎么样来学习做什么。我们假设每个 Agent 总是用一个混合策略来比赛；Agent 根据收到的收益来更新动作的概率。为了简单起见，我们假设一个单一的状态；每次改变的只有其他 Agent 的随机策略。

图 10-9 的 *PolicyImprovement* 算法给学习 Agent 提供了一个控制器。数组 P 中存放着它的当前随机策略，数组 Q 中存放着每个动作的估计收益。Agent 根据它的当前策略来采取动作，并观察动作的收益。然后它更新它的动作值的估计，并且通过增加它的最好动作的概率来改变它的现有策略。

在这个算法中， n 是动作的数目 (A 中元素的数目)。首先，随机初始化 P ，使 P 是一个概率分布； Q 被任意初始化为 0。

在每个阶段，Agent 根据当前的分布 P 选择一个动作 a 。执行 a 并且观察它收到的收益。然后它更新 a 的估计收益。它以学习速率 α 进行梯度下降，以尽量减少动作 a 的估计收益的误差。如果收益超过了它先前的估计，就与误差成比例地增加估计。如果收益低于它的估计，就减少它的估计。

然后根据它估计的 Q 值，计算出一个目前最佳的动作 a_best 。(假设，如果有一个以上的最佳动作，可以将任意一个作为 a_best 。)它通过 $(n-1)\delta$ 来增加最佳动作的概率，通过 δ 来减少其他动作的概率。第 23 行上的 *if* 条件是用来保证概率都是非负的，并且总和为 1。

即使 P 的某个动作概率为 0，也可以偶尔用它来更新当前值。在下面的例子中，我们

```

1: procedure PolicyImprovement( $A, \alpha, \delta$ )
2:   Inputs
3:      $A$ : 动作集
4:      $\alpha$ : 估计动作的步长
5:      $\delta$ : 概率变化的步长
6:   Local
7:      $n$ :  $A$  的元素数量
8:      $P[A]$ :  $A$  上的概率分布
9:      $Q[A]$ : 执行  $A$  的估计值
10:     $a\_best$ : 当前最好的动作
11:     $n \leftarrow |A|$ 
12:     $P[A]$  随机赋值,  $P[a] > 0$  且  $\sum_{a \in A} P[a] = 1$ 
13:     $Q[a] \leftarrow 0$ , 对每一个  $a \in A$ 
14:    repeat
15:      select 动作  $a$  基于分布  $P$ 
16:      do  $a$ 
17:      observe payoff
18:       $Q[a] \leftarrow Q[a] + \alpha(\text{payoff} - Q[a])$ 
19:       $a\_best \leftarrow \arg \max(Q)$ 
20:       $P[a\_best] \leftarrow P[a\_best] + n \times \delta$ 
21:      for each  $a' \in A$  do
22:         $P[a'] \leftarrow P[a'] - \delta$ 
23:        if  $P[a'] < 0$  then
24:           $P[a\_best] \leftarrow P[a\_best] + P[a']$ 
25:           $P[a'] \leftarrow 0$ 
26:    until 终止
  
```

图 10-9 学习协调算法

假设 Agent 每一步以概率 0.05 选择一个随机动作, 否则根据 P 中的动作的概率来选择每个动作。

本书的网站中有一个开放源代码的 Java 小程序, 实现了学习控制器。

【例 10-17】 图 10-10 显示了例 10-12 的学习算法的一个情境。这个图给出运行了 7 次学习算法的 Agent 1 选择足球和 Agent 2 选择购物的相对概率。每一条线是一次运行。每次运行在左上角或右下角结束。在这些运行中, 策略是随机初始化的, $\alpha=0.1$, $\delta=0.01$ 。

如果其他的 Agent 采用一个固定的策略(即使该策略是一个随机策略), 这个算法也将收敛到一个最佳应对(只要 α 和 δ 足够小, 并且 Agent 偶尔随机尝试所有的动作)。

下面的讨论假定所有 Agent 都使用这种学习控制器。

如果在纯策略里有唯一的纳什均衡, 并且所有的 Agent 使用这个算法, 它们将会收敛到这个均衡。劣势策略会将其概率设为零。在例 10-15 中, 存在纳什均衡。同样, 对于例 10-13 中的囚徒困境, 它将会收敛于两个 Agent 都采取“take”行为的唯一均衡。因此, 这个算法没有学习合作(cooperate), 囚徒困境里合作的 Agent 双方都会采取“give”动作以使它们的收益最大化。

如果有多个纯均衡, 这个算法将收敛到其中的一个。因此 Agent 要学习协调(coordi-nate)。在例 10-12 足球赛-购物博弈中, 它将会收敛到两个都去购物或者两个都去看足球赛中的一个均衡。收敛到哪一个取决于初始策略。

如果只有一个随机的均衡, 如例 10-9 罚点球的例子, 这个算法往往在均衡之间循环。

【例 10-18】 图 10-11 显示了两个参赛者对例 10-9 使用学习算法的情况。此图描绘了运行一次学习算法中守门员向右跳和射门者向左踢的相对概率。在这次运行中, $\alpha=0.1$, $\delta=0.001$ 。学习算法在均衡之间循环, 但从来没有真正达到均衡。

考虑 2-Agent 竞争性博弈, 其中只有一个随机的纳什均衡。如果 Agent A 正在和另一个 Agent B 比赛, 其中有一个纳什均衡, Agent A 执行它的支持集中的哪个行为都可以; 因为它们对 A 有相同值。因此, Agent A 往往会偏离均衡。需要注意的是, 当 A 偏离均衡策略时, 对 Agent B 来说最好的回应是确定性地比赛。当 Agent B 最终使用这个算法时, 注意到 A 已经偏离均衡且 Agent B 改变它的策略。Agent B 也将要偏离这个均衡。然后 Agent A 能尝试利用这个偏差。当它们都用这个控制器时, 每个 Agent 的偏差都可以被利用, 而且它们接近于循环。

这个算法不能使 Agent 处于一个随机的均衡。不让 Agent 偏离均衡太远的一个方法是采用赢或快速学习(win or learn fast, WoLF)策略: 当 Agent 赢的时候, 它采取小的步骤

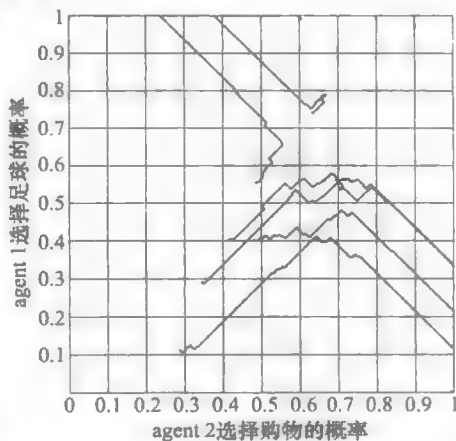


图 10-10 足球购物的学习协调的例子

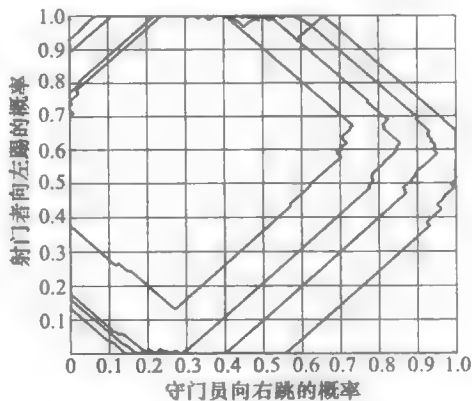


图 10-11 足球罚点球的例子

(δ 是小的); 当 Agent 输的时候, 它采取较大的步骤(δ 是增加的)。当它赢的时候, 它趋向于保持相同的策略, 当它输的时候, 它会尝试迅速采取行动以达到一个更好的策略。为了定义胜利, 对于 Agent 来说一个简单的策略是看它是否比迄今为止它收到的平均收益更好。

要注意的是, 没有完美的学习策略。如果对方 Agent 知道 Agent A 正在使用的准确策略(无论是不是学习的), 并且可以预测 Agent A 将要做什么, 那么它就可以利用这些知识。

10.5 群体决策

通常情况下, 群体的人必须决定该群体将做什么。社团是典型的例子, 投票就是用来确定群体想要什么。看起来投票是一个决定一个群体想要什么的很好的方式, 并且当有一个明确的最优先选择时, 它就是一个好的方式。然而, 当没有一个明确的优先选择时, 投票有很多关键问题, 如下面的例子所示。

【例 10-19】 考虑一个采购 Agent 必须要为一群人根据他们的偏好来决定一个度假目的地。假设有三个人, Alice、Bob 和 Cory, 三个目的地, X、Y 和 Z。假设 Agent 有以下偏好, $>$ 意思为严格地偏好:

- Alice: $X > Y > Z$ 。
- Bob: $Y > Z > X$ 。
- Cory: $Z > X > Y$ 。

给定这些偏好, 在一个投票对中, $X > Y$ 是因为在三人中的两个人更偏好 X。同样, 在投票中, $Y > Z$ 并且 $Z > X$ 。因此, 通过投票得到的偏好是不可传递的。这个例子被称为孔多赛悖论(Condorcet paradox)。事实上, 在这种情况下并不明确群体的结果是什么, 因为它在结果之间是对称的。

445

社会偏好函数(social preference function)为一个群体给出了一个偏好关系。我们希望社会偏好函数依赖于群体中的个人偏好。孔多赛悖论看起来似乎是一个投票对的问题; 但是, 下面的结果表明这种悖论随着任何社会偏好函数发生。

命题 10.1(阿罗不可能定理) 如果有三个或者更多的结果, 任何社会偏好函数不能同时具有以下性质:

- 社会偏好函数是完备的并且是传递的。
- 允许个人具有完备可传递的偏好。
- 如果在个人偏好中有 $o_1 > o_2$, 那么在群体偏好中有 $o_1 > o_2$ 。
- 在结果 o_1 和 o_2 中, 群体偏好仅仅取决于个人对 o_1 和 o_2 的偏好, 而不取决于个人对其他结果的偏好。
- 没有个人可以单方面决定结果(非独裁性)。

当构建一个 Agent 拥有个人偏好并且给出社会偏好, 我们必须清楚我们不可能拥有所有直观的和理想的特性。与其给出一个具有不理想性质的群体偏好, 还不如指出个体偏好间是如何不可调和的。

10.6 机制设计

前面的 Agent 选择动作的讨论中, 假设每个 Agent 都要一个预定义的博弈。机制设计的问题是设计一个博弈, 对于不同的 Agent 有理想的特性。

一个**机制**(mechanism)指定了每个 Agent 可行的动作和每个动作组合的结果分布。我们假设 Agent 在结果上具有效用。

一个机制应具有下面两个常见的理想性质：

- Agent 应该很容易使用机制。给定 Agent 的效用，对 Agent 来说应该很容易决定做什么。无论其他 Agent 做什么，一个**占优策略**(dominant strategy)对 Agent 来说是最好的策略。如果 Agent 有一个占优策略，不需要上一节中的复杂策略推理，它就能做最好的动作。如果每个 Agent 都有一个占优策略，并且在占优策略中 Agent 最好的策略就是声明它的真实偏好，那这个机制就是**真正的**(truthful)占优策略。在一个真正的占优机制中，Agent 只需简单地声明它的真实偏好；Agent 不可能通过操纵机制而得到自己的更好收益。
- 一个机制应该给出聚集了所有 Agent 的最好结果。例如，如果选择的结果能最大化所有 Agent 效用的总和，那这个机制在经济上就是有效的。

446

【例 10-20】 假设你想设计一个会议调度程序，用户输入他们空闲的时间，调度程序选择一个开会的时间。第一种机制是用户指定他们是否空闲的时间，并且调度程序选择大多数人都空闲的时间。第二种机制是用户指定他们不同时间的效用，并且调度程序选择效用总和最大化的时间。这些机制都不是真实的占优策略。

对于第一种机制，用户可以声明他们某些时间是不可行的且不能强制一个他们更偏好的时间。我们并不清楚，一个确定的空闲时间是否是预先定义好的；在某个阶段，用户必须决定他们是否很容易重新安排在一些特定时间要做的其他事情。不同的人更改为其他活动安排可能有不同的阈值。

对于第二种机制，假设有三个人，Alice、Bob 和 Cory，并且他们必须决定是否在周一、周二，或者周三开会。假设对于开会日期他们有以下效用：

	周一	周二	周三
Alice	0	8	10
Bob	3	4	0
Cory	11	7	6

经济有效的结果是在周二开会。然而，如果 Alice 要把周二的评估改成 2，这个机制将会选择周三。因此，Alice 有动机去谎报她的值。诚实不符合 Alice 的利益。

需要注意的是，如果有一个机制，它有占优策略，那就有一个机制，它是真正占优策略。这就是所谓的**启示原理**(revelation principle)。要实现一个真正的占优策略机制，我们原则上可以写一个程序接受来自 Agent 的真正偏好，并且为这个 Agent 的原始机制提供最优输入。从本质上讲，这个程序可以最好地为 Agent 说谎。

事实证明，设计一个合理的占优策略机制是不现实的。只要有三个或者更多的可以选择的结果，唯一占优策略的机制有一个**独裁者**(dictator)；有一个 Agent，它的偏好决定结果。这就是 Gibbard-Satterthwaite 定理。

获得真正占优策略机制的一个方法是引入货币。假设货币可以添加到效用，对于任何两个结果 o_1 和 o_2 来说，每个 Agent 都有一定的(可能是负的)数 d ，这样该 Agent 对结果 o_1 和 $o_2 + d$ 无所谓。通过给 Agent 一个报酬使其接受一个它们不喜欢的结果，或者支付报酬来得到一个它们想要的结果，我们可以保证 Agent 不会通过说谎来获益。

在 **VCG 机制**(VCG mechanism)或者 Vickrey-Clarke-Groves 机制中，Agent 要声明每

447

个结果的值。我们可以选择值的总和最大的结果。Agent 根据它们的参与影响结果的多少来支付。Agent i 的支付为，其他 Agent 选择的的结果的值的总和减去 Agent i 没有参与的其他 Agent 的值的总和。假设 Agent 只关心它们的效用，不关心其他 Agent 的效用或者其他 Agent 的支付，这个 VCG 机制是经济高效的并且是真正占优策略。

【例 10-21】 考虑例 10-20 中的值。假设给定的值可以被解释为美元；例如，Alice 对于在周一还是周二开会并支付 8 美元并不关心（她准备支付 7.99 美元而不是 8.01 美元来把开会从周一移到周二）。给定了这些声明的值，周二被选择作为会议日。如果 Alice 没有参加，周一将会被选中，并且其他的 Agent 有 3 美元的净亏损，所以 Alice 必须要支付 3 美元。然后她的净值为 5 美元，即周二的效用 8 美元减去支付的 3 美元。下表中给出声明、支付和净值：

	周一	周二	周三	支付	净值
Alice	0	8	10	3	5
Bob	3	4	0	1	3
Cory	11	7	6	0	7
总计	14	19	16		

如果 Alice 改变她对周二的评估为 2 美元，考虑下将会发生什么。在这种情况下，周三将会被选择作为会议日，但是 Alice 将有一个新的 2 美元值，却必须支付 8 美元（ $14 - 6$ ——译者注），所以状况将进一步恶化。Alice 不可以通过对机制说谎来获益。 ◀

出售一个物品或者一批物品的一个常见机制是**拍卖**(auction)。出售单一物品的常见拍卖类型是一个加价拍卖。当以前的报价达到时，通过一个预定的增量来不断增加，就有这个项目的当前报价。以当前价格提出购买这个物品被称为出价。对一个特定的价格只有一个人可以竞标。出价最高并且支付相应金额的人将获得该物品。

考虑出售单一物品的 VCG 机制。假设有一些 Agent，他们每人以自己对该物品的估价来出价。最大化收益的结果是这个物品给出价最高的人。如果他们没参加，该物品将给第二高的投标人。因此，根据 VCG 机制，最高的投标人应得到这个项目并且支付第二高的出价，这就是**第二价格拍卖**(second-price auction)。第二价格拍卖等价于（招标增量）有一个加价拍卖，其中人们使用一个代理出价，并有一个 Agent 来把代理出价转换为真正的出价。第二出价拍卖中竞价非常简单，因为 Agent 不必做复杂的策略推理。确定一个胜者及合理的支付也很容易。

448

10.7 本章小结

- 一个多 Agent 系统是由多个可以自主行动并且对结果有他们自身效用的 Agent 组成。结果取决于所有 Agent 的行动。Agent 可以竞争、合作、协调、通信和协商。
- 博弈的策略性形式指出了每个 Agent 给定的控制器的预期结果。
- 对于博弈树来说，博弈的扩展形式通过时间模型化了 Agent 的动作和信息。
- 一个多 Agent 决策网络模型化了概率的依赖性和信息可用性。
- 完备信息博弈可以通过回溯博弈树中的值，或者使用极大极小 $\alpha\beta$ 剪枝搜索博弈树来解决。
- 在部分可观察的领域，有时随机动作是最好的。
- 纳什均衡对每一个 Agent 是一个策略组合，在该均衡中没有 Agent 可以通过单方面偏离这个组合来增加它的效用。
- Agent 可以通过重复博弈学习协调，但是学习一个随机策略是很困难的。

- 通过引入支付，可以设计一种机制实现经济有效的真正的占优策略。

10.8 参考文献及进一步阅读

对于多 Agent 系统的概述，请参阅 Shoham 和 Leyton-Brown[2008]，Stone 和 Veloso[2000]，Wool-dridge[2002]和 Weiss[1999]。Nisan, Roughgarden, Tardos 和 Vazirani[2007]概述了算法博弈论的当前研究前沿。

多 Agent 决策网络基于 Koller 和 Milch[2003]的 MAIDs。

极小 $\alpha\beta$ 剪枝由 Hart 和 Edwards[1961]首次发布。Knuth 和 Moore[1975]和 Pearl[1984]分析了 $\alpha\beta$ 剪枝和其他搜索博弈树的方法。Ballard[1983]讨论了极大极小是如何与机会节点结合的。

深蓝国际象棋由 Campbell、Hoane Jr. 和 Hse[2002]所描述。

博弈的学习和 WoLF 策略基于 Bowling 和 Veloso[2002]。

机制设计由 Shoham 和 Leyton-Brown[2008]，Nisan[2007]和微观经济学教科书，如 Mas-Colell, Whinston 和 Green[1995]所描述。Ordeshook[1986]对群体决策和博弈论有很好的描述。

449

10.9 习题

- 10.1 对于例 10-11 的鹰-鸽博弈，当 $D > 0$ 且 $R > 0$ 时，每个 Agent 试图最大化它的效用。存在一个随机策略的纳什均衡吗？概率是多少？每个 Agent 的预期收益是多少？（这些应该表示为 R 和 D 的函数）。写出你的计算。
- 10.2 在例 10-12 中，什么是随机策略的纳什均衡？在这个均衡中，对于每个 Agent 来说期望值是多少？
- 10.3 在连续的囚徒困境中，假设有一个折扣系数 γ ，这意味着在每个阶段停止的概率是 γ 。对于 γ 的所有值，针锋相对是一个纳什均衡吗？如果是，证明它。如果不是，那 γ 的哪些值是纳什均衡？

450

有监督之外的其他学习模型

学而不思则罔；思而不学则殆。

——孔子(公元前 551—前 479)，《论语》

第 7 章介绍了有监督学习，本章介绍其他的一些学习模型，包括学习更为丰富的表示形式和学习如何行动等问题，这使得学习能够和推理结合在一起。首先介绍无监督学习，其中训练数据没有给定类别信息。然后介绍信念网络学习，无监督学习是信念网络学习的一种特殊情形。最后介绍增强学习，其中的 Agent 在与环境交互的同时学习如何行动。

11.1 聚类

第 7 介绍了有监督学习，其中依据输入特征预测的目标特征在训练数据中是存在的。在**聚类**(clustering)或**无监督学习**(unsupervised learning)中，训练数据没有给定目标特征，目标是构建一个自然的可用于聚集数据的分类。

聚类的基本思想是将样本集划分成**簇**(cluster)或**类**(unsupervised learning)。每个类预测在该类内的样本的特征值。每个聚类都有一个预测误差，最小化误差的那个聚类是最优的。

【例 11-1】 诊断助手可能想将不同的疗法进行分组，使得相同组内的疗法具有令人满意或不满意的治疗效果。诊断助手可能不会为病人提供这样一种药物：与之相似的药物已经对类似的病人造成了可怕的后果。

451

一个智能教学系统可能想根据学习习惯将学生进行聚类，使得对某个成员有效的教学策略也适用于同类内的其他成员。

在**硬聚类**(hard clustering)中，每个样本是明白无误地属于某个类。该类可用于预测样本的特征值。另一种方法是**软聚类**(soft clustering)，其中每个样本在它属于的类上存在一个概率分布。样本特征的预测值是该样本所属类的预测的加权平均值，权值是样本属于某类的概率。

11.1.1 期望最大化

期望最大化(Expectation Maximization, EM)算法可用于聚类。对于给定的数据，EM 学习这样一个原理：它说明如何分类每个样本和预测每个类的特征值。其基本思想是：从随机理论或随机已分类的数据出发，重复下面的两个步骤直至收敛：

E：使用现有的理论分类数据。

M：使用现有的数据分类生成最优的理论。

E 步骤为每个样本生成期望的分类。M 步骤在给定已分类的数据的前提下生成最可能的理论，它是一个监督学习问题。作为一个迭代算法，它可能陷入局部最优；不同的初始值会影响找到的最终理论。

下面两节介绍 EM 算法的两个实例。

11.1.2 k -均值

k -均值算法(k -means algorithm)用于硬聚类。该算法的输入是训练样本和类别的数目 k , 输出是 k 个类的集合、每个类各个特征的预测值和样本与类别的对应关系。

k -均值算法假定特征均为数值型, 目标是当样本的预测值源于样本所属的类别时, 找到最小化平方和误差的类别划分。

假定 E 是样本的集合, 输入特征为 X_1, \dots, X_n 。 $val(e, X_j)$ 是输入特征 X_j 在样本 e 上的值。我们将为每个类分别关联一个整数 $i \in \{1, \dots, k\}$ 。

k -均值算法的输出是:

- 函数 $class: E \rightarrow \{1, \dots, k\}$ 。这意味着 $class(e)$ 是样本 e 所属的类别。如果 $class(e)=i$, 称 e 属于类别 i 。
- 函数 $pval$ 。对于类别 $i \in \{1, \dots, k\}$ 和特征 X_j , $pval(i, X_j)$ 表示类别 i 中的每一个样本在特征 X_j 上的预测值。

452

给定 $class$ 函数和 $pval$ 函数, 平方和误差是:

$$\sum_{e \in E} \sum_{j=1}^n (pval(class(e), X_j) - val(e, X_j))^2$$

我们的目标就是找出最小化平方和误差的 $class$ 函数和 $pval$ 函数。

如命题 7.1 所示, 为了最小化平方和误差, 一个类的预测值应该是属于该类的样本预测的平均值。不幸的是, 存在非常多的将样本划分为 k 个类的方法, 搜索其中最优的划分并非易事。

k -均值算法迭代地减少平方和误差。首先, 它随机地为每个类别赋予样本; 然后执行下面两个步骤:

M: 对于每个类别 i 和特征 X_j , 指定 $pval(i, X_j)$ 为所有 $val(e, X_j)$ 的平均值(e 属于类别 i):

$$pval(i, X_j) \leftarrow \frac{\sum_{e, class(e)=i} val(e, X_j)}{|\{e, class(e)=i\}|}$$

其中分母是属于类别 i 的样本个数。

E: 重新为每个样本分配类别: 将样本 e 分配给最小化下式的类别 i :

$$\sum_{j=1}^n (pval(i, X_j) - val(e, X_j))^2$$

重复上述两个步骤直至第二步不再改变样本的分配。

如果执行 M 步和 E 步不改变样本的分配, 则称这种分配是稳定的(stable)。注意, 稳定分配的类别标签的任意排列仍然稳定的。

k -均值算法最终将收敛到一个稳定的局部最值。这是易于理解的, 因为平方和误差保持减少的趋势和仅存在有限次的分配。该算法常常经过少数几次迭代就收敛。 k -均值算法不能保证收敛到一个全局的最小值。为了改善该算法的结果, 可以尝试不同的初始分配, 多运行几次该算法。

【例 11-2】 Agent 已经观察到了下列 $\langle X, Y \rangle$ 数据对:

(0.7, 5.1), (1.5, 6), (2.1, 4.5), (2.4, 5.5), (3, 4.4), (3.5, 5), (4.5, 1.5),
(5.2, 0.7), (5.3, 1.8), (6.2, 1.7), (6.7, 2.5), (8.5, 9.2), (9.1, 9.7), (9.5, 8.5)

图 11-1a 画出了这些数据点。假定 Agent 想将这些数据聚成两类。

453

在图 11-1b 中, 数据点随机分配给了类: 一类用 + 表示, 另一类用 × 表示。+ 类的均值是 $\langle 4.6, 3.65 \rangle$, × 类的均值是 $\langle 5.2, 6.15 \rangle$ 。

在图 11-1c 中, 各个数据点根据其与两个均值点的距离重新进行了分配。重新分配后, + 类和 × 类的均值分别变为 $\langle 3.96, 3.27 \rangle$ 和 $\langle 7.15, 8.34 \rangle$ 。

在图 11-1d 中, 数据点重新分配给与其最近的均值点所代表的类。这个分配是稳定的, 因为进一步的重新分配不会改变样本的类属关系。

不同的数据点初始分配可能给出不同的聚类结果。另一个可能出现的聚类结果是图中最下面的那些点(Y 值小于 3 的点)组成一个类, 其他点组成另外一个类。

如果类别的数目设为 3, 则该运行该算法会将图中数据划分成三部分: 右上角一类、左中部一类和最下面的一类。

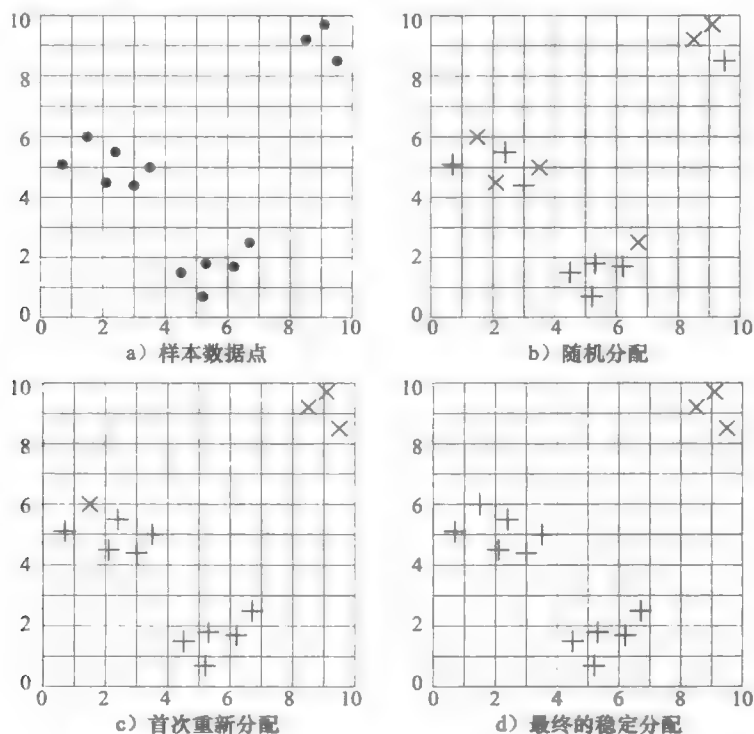


图 11-1 k -均值算法($k=2$)在例 11-2 的数据上的运行过程

k -均值算法的一个问题是需要考虑各维数值的相对比例大小。例如假定有三个特征: *height*、*age* 和一个二元特征, 那么你必须对不同的值域按比例缩放, 以便它们之间可以进行比较。如何缩放也会影响聚类的结果。

Agent 可以通过搜索找出一个合适的类别数目。注意: 只要存在的不同的样本个数大于 k , $k+1$ 个类别总是比 k 个类别具有更小的误差。一个自然的类别数目 k 应该满足这样的条件: 从 $k-1$ 个类别到 k 个类别, 误差的下降非常明显; 而再增加 k , 误差的下降并不明显。此外, 将数据聚为三类的最优划分和聚为两类的最优划分可能是非常不同的。

11.1.3 用于软聚类的期望最大化

EM 算法可以用于软聚类。直观地, 对于聚类而言, EM 算法类似于 k -均值算法, 不同的是前者中的样本是以一定的概率属于某类, 且该概率定义了距离度量。这里假定特征是离散型的。

与 k -均值算法一样，给定了作为输入数据的训练样本和类别的数目。

当进行聚类的时候，类的作用是预测特征的值。为了将 EM 算法用于软聚类，我们采用朴素贝叶斯分类器作为分类的模型，其中，输入特征以一定的概率依赖于类别；在给定的条件下，输入特征之间是相互独立的。类变量有 k 个值，假定为 $\{1, \dots, k\}$ 。

给定朴素贝叶斯模型和训练数据，EM 算法产生分类器所需的概率，如图 11-2 所示。图中的 C 代表类变量。已知类变量的概率分布和给定类别条件下的特征的概率，则可以分类任意的新样本。

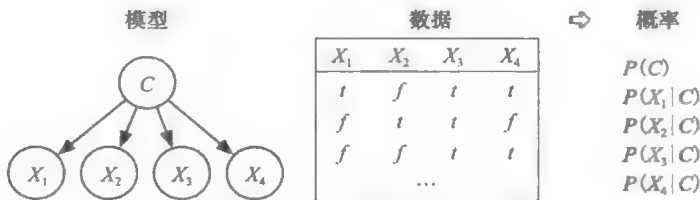


图 11-2 EM 算法：带有隐藏类的贝叶斯分类器

为了初始化 EM 算法，我们用类别特征 C 和计数特征 $count$ 的列扩充数据集。每一个原始的元组均被映射为 k 个元组（每个元组对应一个类）。随机指定这些元组的计数值，使得它们的和为 1。例如，对于具有四个特征和三个类别的数据集，我们可能有以下结果：

X_1	X_2	X_3	X_4	C	Count
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
t	f	t	t	1	0.4
t	f	t	t	2	0.1
t	f	t	t	3	0.5
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots

如果训练样本包含多个具有相同输入特征值的元组，则可将这些元组在扩充数据集中组合在一起。如果训练数据包含这样的 m 个元组，它们的输入特征具有相同的指定值，则在具有这些特征值的扩充数据集中，计数和等于 m 。

如图 11-3 所示的 EM 算法，同时维护概率表和扩充数据集。E 步更新计数值，M 步更新概率值。

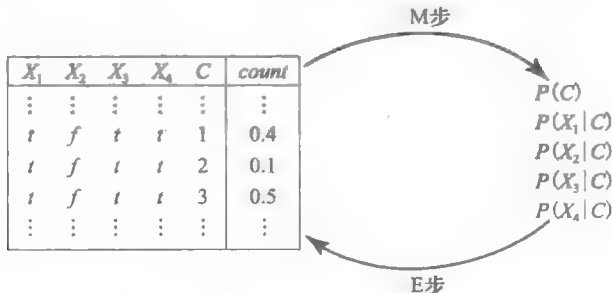


图 11-3 无监督学习的 EM 算法

详细的算法见图 11-4，其中， $A[X_1, \dots, X_n, C]$ 代表扩充数据集； $M_i[X_i, C]$ 是分布 $P(X_i, C)$ 的边缘概率， $P(X_i, C)$ 可由 A 导出； $P_i[X_i, C]$ 代表条件概率 $P(X_i|C)$ 。算法重复下面两个步骤：

- E 步。基于概率分布更新扩充数据集。假定在原始数据集中存在元组 $\langle X_1 = v_1, \dots,$

$X_n = v_n$ 的 m 个备份, 则在扩充数据集中, 与类 c 关联的存储于 $A[v_1, \dots, v_n, c]$ 的计数值更新为:

456

$$m \times P(C = c | X_1 = v_1, \dots, X_n = v_n)$$

注意, 在该步骤中涉及概率推理, 如后所述。

- M 步。由扩充数据集推断模型的最大似然概率。这和从数据中学习概率是一样的问题。

```

1: procedure EM( $X, D, k$ )
2:   Inputs
3:      $X$  特征集  $X = \{X_1, \dots, X_n\}$ 
4:      $D$  具有特征  $\{X_1, \dots, X_n\}$  的数据集
5:      $k$  类别的数目
6:   Output
7:      $P(C), P(X_i | C)$  对于每一个  $i \in \{1: n\}$  其中  $C = \{1, \dots, k\}$ 
8:   Local
9:     实值数组  $A[X_1, \dots, X_n, C]$ 
10:    实值数组  $P[C]$ 
11:    实值数组  $M_i[X_i, C]$ , 对于每一个  $i \in \{1: n\}$ 
12:    实值数组  $P_i[X_i, C]$ , 对于每一个  $i \in \{1: n\}$ 
13:     $s := D$  中的元组数
14:    任意指定  $P[C]$  和  $P_i[X_i, C]$  的值
15:    repeat
16:      for 每一个指派  $\langle X_1 = v_1, \dots, X_n = v_n \rangle \in D$  do
17:        令  $m \leftarrow |\langle X_1 = v_1, \dots, X_n = v_n \rangle \in D|$ 
18:        for 每一个  $c \in \{1: k\}$  do
19:           $A[v_1, \dots, v_n, c] \leftarrow m \times P(C = c | X_1 = v_1, \dots, X_n = v_n)$ 
20:         $P[C] \leftarrow \sum_c A[v_1, \dots, v_n, c]$ 
21:      for each  $i \in \{1: n\}$  do
22:         $M_i[X_i, C] = \sum_{X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_n} A[X_1, \dots, X_n, C]$ 
23:         $P_i[X_i, C] = \frac{M_i[X_i, C]}{\sum_c M_i[X_i, C]}$ 
24:       $P[C] = \sum_{X_1, \dots, X_n} A[X_1, \dots, X_n, C] / s$ 
25:    until 结束
26:

```

图 11-4 无监督学习的期望最大化

上述给出的 EM 算法从随机选择的概率开始运行, 初始的计数也是虚构的。EM 算法将收敛于数据的局部最大似然值。当结果的改变足够小的时候可以终止算法的运行。

457

该算法返回一个可用于分类已存在或新的样本的概率模型。分类一个新样本和计算上述算法第 20 行式子的值的方法是一样的, 即

$$P(C = c | X_1 = v_1, \dots, X_n = v_n) = \frac{P(C = c) \times \prod_{i=1}^n P(X_i = v_i | C = c)}{\sum_{c'} P(C = c') \times \prod_{i=1}^n P(X_i = v_i | C = c')}$$

上述概率是最终学习模型的一部分。

注意上述算法与 k -均值算法的相似性。E 步以一定的概率将样本分配给类, M 步确定类别的预测是什么。

【例 11-3】考虑图 11-3, 假定 E' 是扩充的样本集 (即增加了类变量 C 和计数 $count$ 列) 且存在 m 个样本。因此, E' 中的计数和总是 m 。

在第 M 步中, $P(C=i)$ 被设定为 $C=i$ 的计数的比例:

$$\frac{\sum_{x_1, \dots, x_n} A[x_1, \dots, x_n, C=i]}{m}$$

扫描一遍数据集就可以计算上式的值。

由于 $M_i[x_1, C]$ 可写成

$$\sum_{x_2, x_3, x_4} A[x_1, \dots, x_4, C]$$

因此, 通过扫描一遍数据集更新所有的 $M_i[x_i, C]$ 数组是可能的。见习题 11.3。可以通过将 M_i 数组归一化来获得用 P_i 数组表示的条件概率。

第 E 步更新扩充数据集中的计数值。它将图 11-3 中的 0.4 替换为

$$P(C=1 | x_1 \wedge \neg x_2 \wedge x_3 \wedge x_4) = \frac{P(x_1 | C=1)P(\neg x_2 | C=1)P(x_3 | C=1)P(x_4 | C=1)P(C=1)}{\sum_{i=1}^3 P(x_1 | C=i)P(\neg x_2 | C=i)P(x_3 | C=i)P(x_4 | C=i)P(C=i)}$$

这些概率是最终学习模型的一部分。

注意, 只要 $k > 1$, EM 算法事实上总是具有多个局部最大值。特别的, 一个局部最大值所对应的类标签的任意排列仍将得到一个局部最大值。

11.2 信念网络学习

信念网络给出一个随机变量集的概率分布。我们不能总是期望专家能够提供准确的模型; 通常, 我们想从数据学习一个网络。

从数据中学习一个信念网络能够说明许多问题, 这些问题依赖于知道多少先验信息以及数据集的完整性如何。最简单的情形是: 结构是已知的, 且每个样本中的变量是可见的, 需要学习的仅是概率。另一个极端的情形是: 解释数据的假设变量是未知的, 且存在不完整的数据(数据的丢失不能假定是随机的)。

458

11.2.1 概率学习

如上所述, 最简单的情形是, 我们已经知道模型的结构和已观察到所有的变量, 此时, 我们仅需学习概率。这和 7.3.3 节所介绍的概率学习非常类似。

利用经验数据和伪计数或者依据 Dirichlet 分布, 我们可以单独地学习每一个条件概率分布。

【例 11-4】 图 11-5 显示的是一个典型的情形。给定模型和数据, 目标是推断概率。

例如, $P(E|AB)$ 其中的一个元素是

$$P(E=t | A=t \wedge B=f) = \frac{(\# \text{ examples } E=t \wedge A=t \wedge B=f) + c_1}{(\# \text{ examples } A=t \wedge B=f) + c}$$

其中, c_1 是情形 $E=t \wedge A=t \wedge B=f$ 的伪计数, c 是情形 $A=t \wedge B=f$ 的伪计数。注意, $c_1 \leq c$ 。

如果一个变量拥有多个父节点, 使用计数或伪计数可能导致过拟合。对于父变量节点的一些组合而言, 当仅存在少量样本的时候, 过拟合是最严重的。此时, 可以利用第

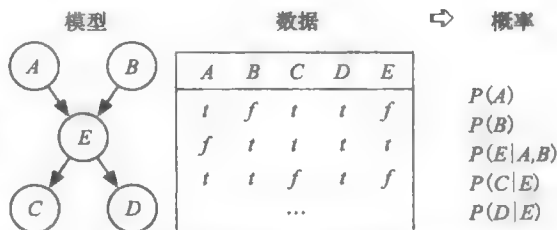


图 11-5 从模型和数据中学习概率

459

7章介绍的一些技术(例如,叶节点为概率的决策树学习、sigmoid 线性函数和神经网络)避免过拟合。在给定变量 X 的父节点条件下,使用有监督学习方法学习 X 的条件概率时,父节点变成了输入节点、 X 变成了目标特征。决策树可用于任意的离散变量。sigmoid 线性函数和神经网络可以表示一个二元变量(给定其父节点)的条件概率。如果不是二元变量,则可使用指示变量。

11.2.2 未观察到的变量

另一种比较简单的情形是:给定了模型,但并非所有的变量是可见的。隐藏变量(hidden variable)或潜在变量(latent variable)指的是在信念网络中未观察到其值的变量,即在数据集中不存在与该变量相对应的列。

【例 11-5】图 11-6 显示的是一种典型的情形。假定所有的变量都是二元变量。模型中包含一个不在数据集中出现的变量 E 。目标是学习包括隐藏变量 E 在内的模型的参数。在该例子中,存在 10 个需要学习的参数。



图 11-6 从缺失数据中学习概率

注意,如果模型忽略变量 E ,则算法必须学习 $P(A)$ 、 $P(B)$ 、 $P(C|AB)$ 和 $P(D|ABC)$,它们总共有 14 个参数。引入隐藏变量的原因就是为了让模型更为简单,因而也就更不易于产生过拟合。

本质上,学习带有隐藏变量的信念网络的 EM 算法(见图 11-7)与用于聚类的 EM 算法是一样的。前者的 E 步可能涉及更复杂的概率推理,因为对于每一个样本,它都需要在给定观察到的变量的条件下,推断隐藏变量的概率。用于聚类的 EM 算法的 M 步从扩充数据集中推断模型的参数,这和前节讨论的可观察到所有变量的情形是一致的;但是在扩充数据集中,计数值不一定是整数。

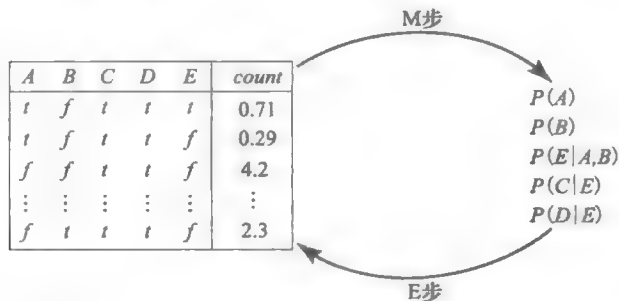


图 11-7 EM 算法用于带有隐藏变量的信念网络

11.2.3 缺失数据

除了未观察到的变量之外,不完全数据也可能是因为其他原因而导致的。数据集的某些元组缺失一些变量的值是可能的。当一些变量的值缺失的时候,我们使用这些数据集就必须十分小心,因为缺失的数据可能和我们感兴趣的现象相关。

【例 11-6】假定存在一种据称对某种疾病有效的治疗方法,其实这种治疗方法对该种疾病根本无效,它只能使病情加重。如果随机地指定病人接受这种治疗,则应该将最严重

460

的病人排除在实验之外, 因为病情太严重致使他们不能参与这项实验。与没有接受这种治疗的病人相比, 接受这种治疗的病人将以更快的速度退出实验。因此, 如果忽略了病人的缺失数据, 则这种治疗看起来好像是有效的。接受这种治疗且仍然没有退出实验的病人会越来越少。

如果数据的缺失是随机的, 则可以忽略这些缺失的数据。然而, “随机缺失”是一个很强的假定。一般的, Agent 会构建一个解释为何缺失数据的模型, 或更可取的是, 它应该去外面的世界调查为什么数据会丢失。

11.2.4 结构学习

如果我们拥有完整的数据且不存在隐藏变量, 但信念网络的结构未知, 这就是信念网络的结构学习(structure learning)要处理的情形。

结构学习存在两种主要的方法:

- 利用基于条件独立性的信念网络的定义。给定变量的一个全序关系, 使得变量 X 的父节点是 X 的前驱变量的子集, 而其他变量与 X 相互独立。这种方法存在两个挑战性的问题: 其一是如何确定最优的全序关系; 其二是要找出一种度量独立性的方法。当只有有限的数据时, 确定条件独立性是不容易的。
- 第二种方法对网络进行评分(例如利用 MAP 模型), 它同时考虑与数据的匹配性和模型的复杂性。给定这样的一个度量, 我们就可以搜索最小化误差的结构。

本节只讨论第二种方法, 通常称之为搜索与评分(search and score)方法。

假定数据是样本集 E , 其中每个样本的变量值均已知。

搜索与评分方法的目标是选择最大化下式的模型:

$$P(model | data) \propto P(data | model)P(model)$$

其中似然性 $P(data | model)$ 是每一个样本概率的乘积。利用积的分解, 给定模型的前提下每一个样本概率的乘积是每一个变量概率的乘积(给定变量父节点的条件下)。因此,

$$\begin{aligned} P(data | model)P(model) &= \left(\prod_{e \in E} P(e | model) \right) P(model) \\ &= \left(\prod_{e \in E} \prod_{X_i} P_{model}^e(X_i | par(X_i, model)) \right) P(model) \end{aligned}$$

其中, $par(X_i, model)$ 是 X_i 的父节点, $P_{model}^e(\cdot)$ 是样本 e 在模型中的概率。

上式的最大化可以通过最大化它的对数形式来实现。当采用对数形式时, 上式变为:

$$\log P(data | model) + \log P(model) = \sum_{e \in E} \sum_{X_i} \log P_{model}^e(X_i | par(X_i, model)) + \log P(model)$$

为了使该方法切实可行, 假定将模型的先验概率分解为每个变量的先验概率, 即每一个变量所代表的局部模型的乘积。令 $model(X_i)$ 是变量 X_i 的局部模型。则目标是最大化下式:

$$\begin{aligned} &\sum_{e \in E} \sum_{X_i} \log P_{model}^e(X_i | par(X_i, model)) + \sum_{X_i} \log P(model(X_i)) \\ &= \sum_{X_i} \left(\sum_{e \in E} \log P_{model}^e(X_i | par(X_i, model)) + \sum_{X_i} \log P(model(X_i)) \right) \\ &= \sum_{X_i} \left(\sum_{e \in E} \log P_{model}^e(X_i | par(X_i, model)) + \log P(model(X_i)) \right) \end{aligned}$$

除非信念网络的无环条件限制了变量的父节点之间的关系, 否则可以通过单独优化每个变量来优化整个模型。然而, 如果给定变量的一个全序关系, 则会面临这样一个分类问题: 在给定变量的前驱节点条件下, 我们想预测每一个变量的概率。为了表示

$P(X_i | \text{par}(X_i, \text{model}))$), 我们可以使用诸如叶节点是概率的决策树(见 7.5.1 节), 或者学习一个挤压线性函数。给定变量前驱节点的评分, 我们就可以在变量的全序上进行搜索以最大化模型的评分。

11.2.5 信念网络学习的一般情形

信念网络学习的一般情形是: 网络结构未知、隐藏变量和缺失数据, 甚至不知道存在什么变量。此时存在两个主要问题: 其一是前面已经讨论过的缺失数据问题; 其二是计算的复杂性问题。虽然存在一个定义明确的搜索空间, 这个空间太大致使尝试所有的变量排序和隐藏变量的组合是不可能的。如果仅仅考虑隐藏变量以简化模型(这看起来是合理的), 则搜索空间虽然是有限的, 但依然太大。

可以选择最优的模型(即具有最大后验概率的模型), 也可以取所有模型的平均值。取所有模型的平均值的方法能给出较好的预测, 但对于一个必须理解或证实模型的人来讲, 这种方法是不易解释的。

组合上述方法与缺失数据是一个更困难的问题, 它需要更多的领域知识。

11.3 增强学习

设想在现实世界中行动的一个机器人, 它接收奖赏或惩罚, 并由此决定应该做什么。这就是增强学习(reinforcement learning)的问题。本章只讨论所有变量均可见的单个 Agent 增强学习(虽然 10.4.2 节介绍了多 Agent 增强学习的一种简单形式)。

可以用马尔可夫决策过程形式化增强学习, 但是 Agent 开始的时候只知道可能的状态集和行动集。因此, 状态转移函数 $P(s' | a, s)$ 和奖赏函数 $R(s, a, s')$ 初始时都是未知的。Agent 能够在现实世界中行动, 而且在每一步行动之后, 能够观察世界的状态和获得什么奖赏。假定 Agent 行动的目的是依据折扣因子 γ 获得最优的折扣。

【例 11-7】 考虑如图 11-8 所示的微型增强学习问题, 其中, Agent 可能处于的状态有 6 个, 记为 s_0, \dots, s_5 ; Agent 具有 4 种行为, 即 *UpC*、*Up*、*Left* 和 *Right*。这些是 Agent 在开始行动之前知道的所有信息。Agent 不知道状态是如何设置的、行动的对象是什么以及如何赢得奖赏。

图 11-8 显示了 6 个状态的设置情况。假定行动的含义如下:

upC (表示 up carefully): Agent 向上走, 除了在状态 s_4 和 s_5 时静止不动(获得奖赏值 -1)。

right: 在状态 s_0 、 s_2 和 s_4 时, Agent 向右走, 获得奖赏值 0; 在其他状态时, Agent 静止不动, 获得奖赏值 -1。

left: 在状态 s_1 、 s_3 和 s_5 时, Agent 向左走。在状态 s_0 时, Agent 静止不动, 获得奖赏值 -1。在状态 s_2 时, Agent 静止不动, 获得奖赏值 -100。在状态 s_4 时, Agent 走向 s_0 , 获得奖赏值 10。

up: 除非奖赏值是 0, 否则 Agent 以 0.8 的概率执行与 *upC* 相同的行动, 以 0.1 的概率执行与 *left* 相同的行动, 以 0.1 的概率执行与 *right* 相同的行动。

假定存在一个折扣率是 0.9 的折扣, 这可以解释为在任意步骤中 Agent 离开游戏的可能性是 0.1, 或者解释为 Agent 更喜欢即时而非未来的奖赏。

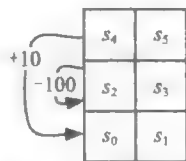


图 11-8 一个微型增强学习问题的环境

【例 11-8】图 11-9 显示了一个更为复杂的游戏。Agent 可以位于 25 个网格中的任意一个，但只有位于角落网格时才有奖赏。当 Agent 位于角落网格并获得奖赏(奖赏值是 10)后，该网格中的奖赏值就会消失。如果网格中没有奖赏，则在每一步骤后均以某个概率在其中的一个角落网格中出现奖赏。怪物可以在任意时刻出现在标为 M 的网格中。如果怪物出现在 Agent 所在的网格，则 Agent 会受到损害，并获得 -10 的奖赏值。Agent 可以通过访问修理站 R 进行修复。

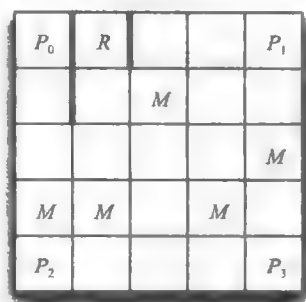


图 11-9 网格游戏的环境

在这个例子中，状态由 4 个元素构成： $\langle X, Y, P, D \rangle$ ，其中 X 和 Y 是 Agent 所处位置的横坐标和纵坐标， P 是具有奖赏值的位置(如果 P_0 有奖赏，则 $P=0$ ；如果 P_1 有奖赏，则 $P=1$ ； $P=2$ 和 $P=3$ 以此类推。如果没有奖赏，则 $P=4$)， D 是布尔变量，当 Agent 受到损坏时 D 取真。由于怪物是转瞬即逝的，因此不必包含在状态的表示之中。状态的总数是 $5 \times 5 \times 5 \times 2 = 250$ 。由于环境是完全可见的，因此 Agent 知道自己处于什么样的状态；但是它不知道状态的含义，开始时也没有关于损坏和奖赏是什么的概念。

Agent 有 4 种行为：向上、向下、向左和向右。这些行为使得 Agent 移动一步(通常移动的方向如行为的名称所示，但有时也会朝其他方向移动)。如果 Agent 撞上外墙或内墙(位置 R 旁边的粗线)，它就停在原位置并获得奖赏值 -1。

Agent 只知道存在 250 个状态和 4 个行为，每一时间它处于什么状态，每次它获得了什么奖赏。除此以外的信息它一概不知。

这是一款简单的游戏，但为它写一个控制程序是非常困难的。本书配套的网站上提供了一个 Java 小程序，用于参考和修改。请尝试自己写一个这样的控制程序；写一个“每运行一千步平均获得 500 奖赏值”的控制程序是可能的。学习该游戏也是不容易的，因为只有直到 Agent 意识到损坏是不好的和访问 R 可以修复损坏，访问 R 看起来才有意义。Agent 必须在努力收集奖赏的同时跌跌撞撞地走到 R 进行修复。没有奖赏的状态不会持续太久。此外，Agent 必须在没有给定“损坏”的概念的条件下进行学习；它初始时只知道存在 250 个状态和 4 种行为。

增强学习是困难的，原因如下：

- **责任归属问题**(blame attribution problem)指的是确定哪个行为对奖赏或惩罚负责。该行为可能在获得奖赏很久以前就发生了。此外，往往不是单个行为而是在适当的环境下多个行为的组合才获得了奖赏。例如，可以通过比输赢训练 Agent 玩游戏；Agent 必须确定想获得胜利采取什么行动才是明智的。你可以尝试训练一条狗：当你回家时发现家里一片狼藉就骂“坏狗”。狗必须确定，在它先前的所有行为当中，哪些导致了主人的训斥。
- 即使状态没有发生改变，Agent 行为的效果也是依赖于 Agent 未来将做什么。那些开始时看似不好的行为有可能最后是最优的，这是因我们并不知道 Agent 将来要做什么。这种现象在规划问题中是常见的，但在增强学习中显得较为复杂，因为 Agent 预先并不知道行为的后果。
- “探索与利用”两难问题：如果 Agent 已经学得了一个优秀的行动方案，它是继续这种方案(利用已经确定的行为)，还是探索更好的方案？不再进行探索的 Agent 将沿着先前得到的方案继续前行，而总是进行探索的 Agent 不会利用学到的知识。该两难问题将在 11.3.4 节作进一步讨论。

11.3.1 演化算法

求解增强学习的一种方法是将它视为一个最优化问题，目标是选择一个最大化期望总奖赏的策略。可以通过策略搜索(policy search)的方法解决该优化问题，它在所有策略构成的空间中进行搜索以找到最好的策略。策略是一个控制程序，可以在 Agent 活动环境中运行该程序以评估其性能。

策略搜索的做法通常是作为一个随机局部搜索算法在策略空间中进行搜索。可以通过在真实环境中运行多次来评估策略的好坏。

策略搜索的一个难点是如何表示策略。从初始策略开始，然后在真实环境中反复评估该策略并迭代地提升性能。这个过程称为演化算法(evolutionary algorithm)，因为作为一个整体的 Agent 的评估依赖于其生存状况。演化算法往往和基因算法组合在一起，基因算法使得我们能更进一步地与生物的进化作类比：Agent 之间相互竞争会导致基因突变。基本思想是交叉选择提供了一种组合策略最佳特性的方法。

演化算法存在一些问题。首先是状态空间的大小。如果有 n 个状态和 m 个行动，则有 m^n 种策略。例如，例 11-7 描述的游戏中有 $4^6 = 4096$ 种不同的策略；例 11-8 描述的游戏中有 250 个状态，因此有 $4^{250} \approx 10^{150}$ 种策略。这是一个非常小的游戏，但它可能的策略数却超过了宇宙中所有粒子的数目。

第二，演化算法以一种极其浪费的方式利用经验知识。如果一个 Agent 位于例 11-7 所示的状态 s_2 ，并且移向左边，此时你希望它学得这样的知识：从状态 s_2 向左走是不对的。但是演化算法一直要等到 Agent 已经从整体上完成和评判该策略之后才能获得这样的知识。随机局部搜索将在状态 s_2 随机尝试其他行动，因此虽然可能最终判定那个行动是不对的，但这是非常不直接的。基因算法比演化算法要好一点，因为 Agent 在状态 s_2 向左走的策略会导致 Agent 死亡，但依然是非常不直接的。

466

第三，演化算法的性能对策略的表示形式非常敏感。基因算法中策略的表示应该使得交叉选择将策略中好的部分保留下来。策略的表示通常是随着不同的特定领域而调整的。

本章的剩余部分将讨论“一步一学习”的方法，即不是从整体上学习一个策略，而是学习策略的各个组成部分。通过在每一个状态中学习做什么，我们能使问题的规模控制在状态数目的线性范围内，而非状态数目的指数范围。

11.3.2 时间差

为了理解增强学习是如何工作的，首先考虑如何调和按顺序提供给 Agent 的经验信息。

假定存在一系列的数值： v_1, v_2, v_3, \dots ，目标是在给定前面数值的条件下预测下一个数值。一种方法是取 v 期望值的一个运行逼近。例如，给定一系列学生的成绩，下一个学生的成绩的合理预测是前面这些成绩的平均值。

令 A_k 是基于前面 k 个数据点 v_1, \dots, v_k 的期望值的估计。一个合理的估计是取平均值：

$$A_k = \frac{v_1 + \dots + v_k}{k}$$

由此，

$$kA_k = v_1 + \dots + v_{k-1} + v_k = (k-1)A_{k-1} + v_k$$

两边同时除以 k ，得到：

$$A_k = \left(1 - \frac{1}{k}\right)A_{k-1} + \frac{v_k}{k}$$

令 $\alpha_k = \frac{1}{k}$, 则

$$A_k = (1 - \alpha_k)A_{k-1} + \alpha_k v_k = A_{k-1} + \alpha_k (v_k - A_{k-1}) \quad (11.1)$$

差 $v_k - A_{k-1}$ 称为时间差分误差(temporal difference error)或 TD 误差, 它说明了新的值 v_k 与旧的预测值 A_{k-1} 之间的差异有多大。通过将 α_k 乘以 TD 误差来更新旧的预测值 A_{k-1} , 并得到新的估计值 A_k 。时间差分误差的定性解释是: 如果新的值大于旧的预测值, 则增加预测值; 反之, 如果新的值小于旧的预测值, 则减小预测值。增加或减少的量和新的值与旧的预测值之间的差成正比。注意, 该公式对于第一个值($k=1$)仍然成立。

上述分析假定所有的值具有相同的权重。然而, 考虑对学生成绩的期望值做保守的估计: 如果学校开始时给的成绩就比较高, 则新的值比旧的成绩对于目前的估计更有利, 因此新的值应该给予更大的权重。

467

在增强学习中, 后面的值 v_k (即更近的值) 比前面的值更准确, 应该给予更大的权重。对后面的样本加更大的权重的一种方法是使用式(11.1), 不同的是, 这里的 α ($0 < \alpha \leq 1$) 是一个不依赖于 k 的常数。不幸的是, 当序列中存在易变的值的时候, 它不能收敛到平均值。虽然如此, 如果潜在的过程致使数值发生变化, 它能够追踪这种变化。

可以采用缓慢地减少 α 值的方法获得两方面的益处: 对新近的样本加更大的权重且仍然收敛于平均值。如果满足下列条件则可保证收敛:

$$\sum_{k=1}^{\infty} \alpha_k = \infty \text{ 且 } \sum_{k=1}^{\infty} \alpha_k^2 < \infty$$

前一个条件确保随机函数和初始条件排除了平均值的情况, 后一个条件保证收敛性。

注意, 当潜在的过程使得生成的值不断发生变化时, 保证收敛到平均值和能够调整到做更优的预测是不相容的。

在本章的后续部分, 没有下标的 α 假定是一个常数; 带有下标的 α 是样本个数的函数, 这些样本的组合用于特定的估计。

11.3.3 Q-学习

在 Q-学习及其相关算法中, Agent 试图从它与环境的交互历史中学习最优的策略。Agent 的历史(history)是一个“状态—行为—奖赏”序列:

$$\langle s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, r_4, s_4, \dots \rangle$$

意思是: Agent 在状态 s_0 执行了行动 a_0 , 获得奖赏 r_1 和到达状态 s_1 ; 然后执行行为 a_1 , 获得奖赏 r_2 和到达状态 s_2 , 以此类推。

我们将历史的交互信息看做经验的一个序列, 经验(experience)定义为元组:

$$\langle s, a, r, s' \rangle$$

意思是: Agent 在状态 s 执行了行动 a , 获得奖赏 r 和到达状态 s' 。Agent 能够从这些经验数据中学习做什么。和决策理论规划一样, Agent 的目标是最大化某个量, 这通常是折扣奖赏。

468

回忆一下, $Q^*(s, a)$ 表示在状态 s 执行行动 a 的期望值(累积折扣), 然后依据最优的策略执行下一步行动。

Q-学习(Q-learning)利用时间差估计 $Q^*(s, a)$ 的值。在 Q-学习中, Agent 维护一个表 $Q[S, A]$, 其中 S 和 A 分别是状态和行为的集合。 $Q(s, a)$ 表示 $Q^*(s, a)$ 的当前估计值。

一条经验 $\langle s, a, r, s' \rangle$ 为计算 $Q(s, a)$ 提供了一个数据。这个数据就是 Agent 接收的未来值 $r + \gamma V(s')$, 其中 $V(s') = \max_{a'} Q(s', a')$; 它等于目前的奖赏值与未来值的折扣估计的和。新的数据称为返回值。Agent 可以利用时间差公式(11.1)更新 $Q(s, a)$ 的估计值:

$$Q[s, a] \leftarrow Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$$

或等价的形式,

$$Q[s, a] \leftarrow (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$$

图 11-10 显示了 Q -学习控制程序。它假定 α 取固定值; 如果 α 是变化的, 则对于每一个“状态-行为”对的计算是不同的, 算法也必须记录这种计算过程。

469

```

1: controller  $Q$ -learning( $S, A, \gamma, \alpha$ )
2:   Inputs
3:      $S$  是状态的集合
4:      $A$  是行为的集合
5:      $\gamma$  是折扣率
6:      $\alpha$  是步长
7:   Local
8:     实值数组  $Q[S, A]$ 
9:     先前状态  $s$ 
10:    先前行动  $a$ 
11:    随机初始化  $Q[S, A]$ 
12:    观察目前状态  $s$ 
13:    repeat
14:      选择和执行一个行动  $a$ 
15:      观察奖赏  $r$  和状态  $s'$ 
16:       $Q[s, a] \leftarrow Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$ 
17:       $s \leftarrow s'$ 
18:    until 结束

```

图 11-10 Q -学习控制程序

只要任意的状态 Agent 尝试一个行为的次数不受限制(即 Agent 在一个状态不会总是执行相同的行动子集), 则不管 Agent 真正依据的策略是哪个(即对于任意的状态 s , 无论执行哪个行动 a), Q -学习算法都学习一个最优的策略。正因为如此, 称这种 Q -学习为高策略(off-policy)方法。

【例 11-9】考虑图 11-8 中给出的例 11-7。假定这里有一个经验 $\langle s, a, r, s' \rangle$ 的序列和更新值(保留两位小数)(其中, $\gamma = 0.9$, $\alpha = 0.2$; 所有的 Q 值都初始化为 0):

s	a	r	s'	更新值
s_0	upC	-1	s_2	$Q[s_0, upC] = -0.2$
s_2	up	0	s_2	$Q[s_2, up] = -0$
s_4	left	10	s_0	$Q[s_4, left] = 2.0$
s_0	upC	-1	s_2	$Q[s_0, upC] = -0.36$
s_2	up	0	s_4	$Q[s_2, up] = 0.36$
s_4	left	10	s_0	$Q[s_4, left] = 3.6$
s_0	up	0	s_2	$Q[s_0, upC] = 0.06$
s_2	up	-100	s_2	$Q[s_2, up] = -19.65$
s_2	up	0	s_4	$Q[s_2, up] = -15.07$
s_4	left	10	s_0	$Q[s_4, left] = 4.89$

注意奖赏值 -100 和其他奖赏值是如何调和的。在接收到奖赏值是一 100 的经验之后, $Q[s_2, up]$ 的值为

$$0.8 \times 0.36 + 0.2 \times (-100 + 0.9 \times 0.36) = -19.65$$

在下一步, Agent 执行相同的行为但输出结果不同, 此时 $Q[s_2, up]$ 的值变为

$$0.8 \times (-19.65) + 0.2 \times (0 + 0.9 \times 3.6) = -15.07$$

获得在状态 s_2 执行 up 行动的更多经验且没有接收到奖赏值 -100 之后, 较大的负奖赏值将最终被正奖赏值所调和, 因而对 $Q[s_2, up]$ 取值的影响就降低了。如果在状态 s_2 执行 up 行动且重新接收到奖赏值 -100, 则又开始新一轮循环。

当先前的估计值比最近的估计值更糟糕的时候, 考虑 α_k 是如何调和奖赏值是有益的。下面的一个例子显示了确定的行为序列的效果。注意, 当一个行为是确定的, 我们可以使用 $\alpha=1$ 。

【例 11-10】考虑例 11-7。假定 Agent 有下述经验:

$(s_0, right, 0, s_1, upC, -1, s_3, upC, -1, s_5, left, 0, s_4, left, 10, s_0)$

并多次重复这个行为序列。(注意, 一个实际的 Q-学习 Agent 不会持续重复相同的动作, 特别是当如本例所描述的 Q-学习记录的时候, 其中一些值看起来并不好, 但我们认为可以帮助手解 Q-学习是如何工作的。)

图 11-11 显示了多次重复执行这个动作序列时 Q 值是如何更新的。上述表中的 Q 值均初始化为 0。

迭代次数	$Q[s_0, right]$	$Q[s_1, upC]$	$Q[s_3, upC]$	$Q[s_5, left]$	$Q[s_4, left]$
1	0	-1	-1	0	10
2	0	-1	-1	4.5	10
3	0	-1	0.35	6.0	10
4	0	-0.92	1.36	6.75	10
10	0.03	0.51	4	8.1	10
100	2.54	4.12	6.82	9.5	11.34
1000	4.63	5.93	8.46	11.3	13.4
10 000	6.08	7.39	9.97	12.83	14.9
100 000	7.27	8.58	11.16	14.02	16.08
1 000 000	8.21	9.52	12.1	14.96	17.02
10 000 000	8.96	10.27	12.85	15.71	17.77
∞	11.85	13.16	15.74	18.6	20.66

a) 一个确定行为序列(每一个“状态-行为”对都有独立的 α_k 值, $\alpha_k=1/k$)的 Q-学习

迭代次数	$Q[s_0, right]$	$Q[s_1, upC]$	$Q[s_3, upC]$	$Q[s_5, left]$	$Q[s_4, left]$
1	0	-1	-1	0	10
2	0	-1	-1	9	10
3	0	-1	7.1	9	10
4	0	5.39	7.1	9	10
5	4.85	5.39	7.1	9	14.37
6	4.85	5.39	7.1	12.93	14.37
10	7.72	8.57	10.64	15.25	16.94
20	10.41	12.22	14.69	17.43	19.37
30	11.55	12.83	15.37	18.35	20.39
40	11.74	13.09	15.66	18.51	20.57
∞	11.85	13.16	15.74	18.6	20.66

b) 一个确定行为序列($\alpha=1$)的 Q-学习

迭代次数	$Q[s_0, right]$	$Q[s_1, upC]$	$Q[s_3, upC]$	$Q[s_5, left]$	$Q[s_4, left]$
∞	19.5	21.14	24.08	27.87	30.97

c) 全面探索并收敛之后的 Q 值

图 11-11 例 11-10 描述的 Q-学习的过程。Q-学习算法的某次特定运行的更新值

471

a) 每一个“状态-行为”对都有独立的 α_k 值。注意，第一次迭代只更新了即时奖赏值。第二次迭代对正奖赏值做了单步备份。由于存在另外一个 Q 值是 0 的行为，所以没有备份 -1 。第三次迭代做了两步备份。 $Q[s_3, upC]$ 得以更新的原因是在两步之前它的奖赏值是 10。 $Q[s_3, upC]$ 取其经验的平均值： $(-1+(-1)+(-1+0.9 \times 6))/3$ 。

b) $\alpha=1$ ，因此只考虑目前的估计。同样，每次迭代都对奖赏值做了单步备份。在第三次迭代中， $Q[s_3, upC]$ 得以更新的原因是在两步之前它的奖赏值是 10；但由于此时 $\alpha=1$ ，所以算法忽略了先前的估计值，只使用新的经验，即 $-1+0.9 \times 0.9$ 。 $\alpha=1$ 比 $\alpha_k=1/k$ 具有更快的收敛速度，但仅当行为是确定的时候 $\alpha=1$ 才收敛，原因是它隐含地假定最后的奖赏值和最终的状态代表了未来的奖赏值或状态。

c) 图 11-11c 列出了如果算法允许 Agent 进行探索(这是正常现象)，则收敛之后的部分 Q 值。注意，由于存在随机的行为，为了收敛 α 不能为 1；此时的 Q 值比确定行为序列的情形大的原因是这些行为没有形成一个最优的策略。

收敛之后的最终策略是：在状态 s_0 执行动作 up ，在状态 s_2 执行动作 upC ，在状态 s_1 和 s_3 执行动作 up ，在状态 s_4 和 s_5 执行动作 $left$ 。

你可以运行该例子的小程序(在本书配套的网站上可以找到)，并尝试不同的初始值和不同大小的 α 。

11.3.4 探索与利用

Q -学习算法没有指定 Agent 实际上应该做什么。Agent 学习一个可用于确定最优行为的 Q 函数。下列两件事是值得 Agent 去做的：

- 利用(exploit)它已经发现的关于目前状态 s 的知识，该知识是通过执行能够最大化 $Q[s, a]$ 的行为 a 而获得的。
- 探索(explore)的目的是建立当前最优 Q 函数的一个更好的估计。换句话说，它应该选择一个与当前看来是最优的行为不同的行为。

存在许多在探索和利用之间进行折衷的建议方法：

- ϵ -贪心策略以 $1-\epsilon$ 的概率选择贪心的行为(最大化 $Q[s, a]$ 的行为)，以 ϵ 的概率选择随机的行为，其中 $0 \leq \epsilon \leq 1$ 。随着时间的推移改变 ϵ 的大小是可能的。直观的，在算法运行的早期 Agent 应该选择一个比较随机的策略，以促进初始探索；随着时间的推移，它应该选择越来越贪心的策略。
- ϵ -贪心策略的一个问题是它同等看待所有的行为(最优的行为除外)。如果存在两个看起来较好的行为，以及多数看起来没什么希望的行为，则在较好的行为中做选择可能是更明智的：花更多的精力在这些有希望的行为中确定哪个是最优的，而不是在那些没有希望的行为中浪费时间。一种办法是以一定的概率选择行为 a ，这个概率依赖于 $Q[s, a]$ 的值。这就是所谓的“软最大化”(soft-max)行为选择法。通常的方法是利用 Gibbs 或波尔兹曼分布(Boltzmann distribution)，其中在状态 s 选择行为 a 的概率与 $e^{Q[s, a]/\tau}$ 成正比。具体的，在状态 s Agent 选择行为 a 的概率是

$$\frac{e^{Q[s, a]/\tau}}{\sum_a e^{Q[s, a]/\tau}}$$

其中 $\tau > 0$ 是温度参数，它指定应该选择怎样随机的值。当 τ 较大时，任意行为被选中的概率几乎一样；减少 τ 值，则拥有最大值的行被选中的可能性更大；当 $\tau \rightarrow 0$ ，则总是选择最优的行为。

472

- 另一种方法是“面对不确定性的乐观处理”：初始化 Q 函数使得能够促进探索。如果 Q 函数的初始值较大，则未探索的区域看上去也是不错的，以致贪心搜索倾向于进行探索。虽然这促进了探索，但是 Agent 会好长一段时间处于某些“状态-行为”对是不错的幻觉，即使没有证据表明这一点。一个状态看上去很糟糕，仅当它的所有行为看上去很糟糕；但是当所有的行为表明这个状态是不错的时候，获得实际值已经花费了很长时间。换言之， Q 函数旧的估计值可能是十分糟糕的，但却可能保留很长一段时间。为了提高收敛速度，初始值应该和最终值尽量接近；一个过高的估计会降低收敛的速度。如果改变了状态转移函数，则仅仅依赖“面对不确定性的乐观处理”是没有用的，因为它将初始的时间段视作探索的时间，初始探索过后，探索就越来越少了。

研究探索策略和更新 α 的方法之间的相互影响是有趣的，见习题 11.8。

11.3.5 增强学习算法的评估

一个增强学习算法的优劣可以通过考察 Agent 在现实世界中找到怎样的策略和获得多少奖赏来评判。更重要的是，它取决于 Agent 将如何被部署应用。如果在部署之前 Agent 有充足的时间进行安稳的学习，则最终的策略可能是最重要的。如果 Agent 必须在部署的同时进行学习，那么它可能永远也不会到达“学得最优策略”的阶段；在学习过程中所接收的奖赏可能就是 Agent 欲最大化的目标。

展示一个增强学习算法性能的一种方法是用图表画出随迭代步数的变化而变化的累积奖赏值（至今为止接收到的奖赏值的和）。如果一种算法的曲线总是在另一种算法的曲线之上，则说明前者更优。

【例 11-11】 图 11-12 比较了在例 11-8 所示的游戏上的 4 种 Q -学习算法的性能。该图的生成使用了本书配套网站上的小程序的“跟踪控制台”功能。

473

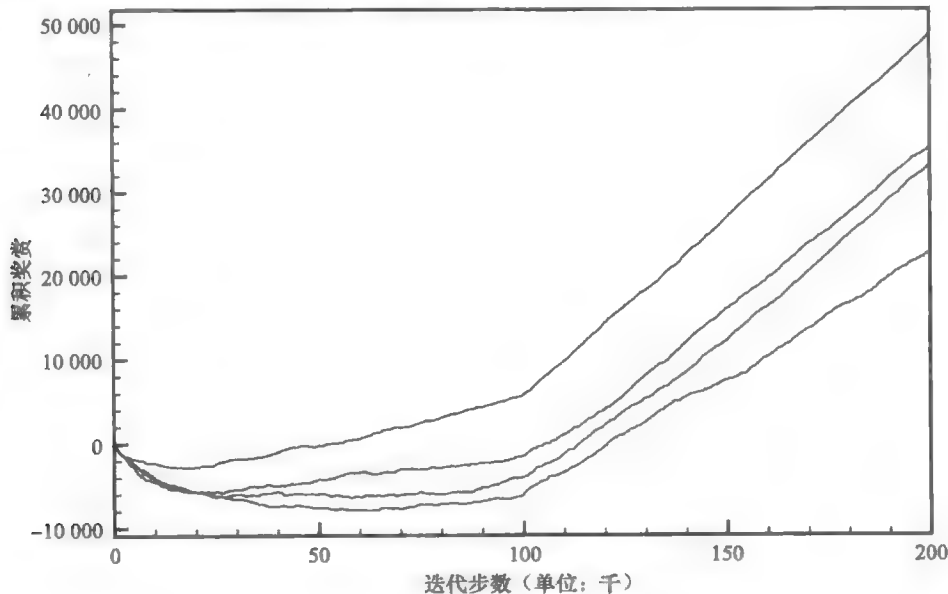


图 11-12 随着迭代步数的增加而变化的累积奖赏值

图上的 4 种不同的 Q -学习由 3 个方面规定： α 是否取固定值、 Q 函数的初始值和行为

选择的随机程度。它们在开始的 100 000 迭代步中,以 80%(即 $\epsilon=0.2$)的概率贪心利用学得的知识;在后续的 100 000 迭代步中,这种概率则为 100%(即 $\epsilon=0$)。图中最上面的算法性能最好。

对一个算法而言,不同次运行的结果会有很大的差异,因此为了准确地比较这些算法,我们必须多次运行相同的算法。在这个例子中,累积奖赏值取决于 Agent 是否学得了访问修理站的知识(Agent 并不总是学习这些知识)。因而这个例子中的累积奖赏值倾向于具有“双峰”的分布,见习题 11.8。

图中有 3 个重要的统计量:

- 渐近斜率说明算法稳定后策略的优劣。
- 曲线的最小点表示累积奖赏值开始增加之前需要牺牲多少奖赏。
- 零交叉点表示算法需要多长时间补偿学习的代价。

当同时存在正奖赏和负奖赏的时候,后面的两个统计量是很有用的;合理的行为能保持正奖赏和负奖赏之间的平衡。对于其他情形,累积奖赏值应该与适合于该例子的合理行为进行比较,见习题 11.7。

关于累积奖赏曲线图必须注意的一个事实是,它度量总的奖赏值;然而算法在每一步优化的是折扣奖赏值。一般的,应该优化最适合于特定问题的优化准则以及利用该准则评估算法的优劣。

11.3.6 在策略学习

只要 Agent 进行了足够的探索,则不管它做什么, Q -学习算法总是学习一个最优的策略。可能存在这样的情形:忽略 Agent 真正做什么的危险的(会有很大的负奖赏值)。一种方法是学习 Agent 即将执行的策略的值,以便能迭代地提升性能。因此,学习器能够考虑与探索相关的代价。

离策略学习器(off-policy learner)学习独立于 Agent 行为的最优策略值。 Q -学习就是一种离策略学习器。在策略学习器(on-policy learner)学习 Agent 即将执行的策略的值,包括探索的步数。

SARSA(这个名字源于它运用“状态-行为-奖赏-状态-行为”经验来更新 Q 值)是一个在策略增强学习算法,它估计即将执行的策略的值。SARSA 中经验的形式为 $\langle s, a, r, s', a' \rangle$,意思是:Agent 在状态 s , 执行行为 a , 接收奖赏 r , 结束于状态 s' , 并由此决定执行行为 a' 。该经验提供了一个用以更新 $Q(s, a)$ 的新值,即 $r + \gamma Q(s', a')$ 。

图 11-13 给出了 SARSA 算法。

SARSA 考虑了目前的探索策略,该策略可能是贪心的(关于随机步骤)。当探索可能导致比较大惩罚的时候, SARSA 可以找

```

controller SARSA( $S, A, \gamma, \alpha$ )
Inputs:
     $S$  是状态的集合
     $A$  是行为的集合
     $\gamma$  是折扣率
     $\alpha$  是步长
Internal State:
    实值数组  $Q[S, A]$ 
    前一状态  $s$ 
    前一行动  $a$ 
begin
    随机初始化  $Q[S, A]$ 
    观察目前的状态  $s$ 
    利用一个基于  $Q$  的策略选择  $a$ 
    repeat forever:
        执行行为  $a$ 
        观察奖赏  $r$  和状态  $s'$ 
        利用一个基于  $Q$  的策略选择  $a'$ 
         $Q[s, a] \leftarrow Q[s, a] + \alpha(r + \gamma Q[s', a'] - Q[s, a])$ 
         $s \leftarrow s'$ 
         $a \leftarrow a'$ 
    end-repeat
end

```

图 11-13 SARSA: 在策略增强学习

到一个与 Q -学习不同的策略。例如，当一个机器人爬到接近楼梯顶的时候，虽然这是一个最优的策略，此时进行探索可能是危险的。SARSA 将发现这种危险，并采用使机器人离开楼梯的策略。它能够在考虑策略内在探索的同时找到最优的策略。

【例 11-12】 在例 11-10 中，最优的策略是在状态 s_0 执行 up 动作(图 11-8)。然而，Agent 进行探索并非好事，因为从状态 s_2 开始探索是非常危险的。

如果 Agent 执行包括探索的策略，“在状态 s ，以 80% 的概率选择最大化 $Q[s, a]$ 的动作 a ，以 20% 的概率随机选择一个动作”，在状态 s_0 执行 up 动作并非是最优的。在策略学习器努力优化 Agent 即将执行的策略，而非不包含探索的最优策略。

假设你重做如图 11-11 所示的实验，则 SARSA 将备份 -1 值，而 Q -学习则不会，原因是存在估计值是 0 的行为。图 11-11a 和图 11-11b 中的 Q 值将收敛于相同的策略值。

与 Q -学习相比，SARSA 中的最优策略值更小。SARSA 中的对应于图 11-11c 的值如下：

迭代步数	$Q[s_0, right]$	$Q[s_1, upC]$	$Q[s_3, upC]$	$Q[s_5, left]$	$Q[s_4, left]$
∞	9.2	10.1	12.7	15.7	18.0

使用 SARSA 算法的最优策略是在状态 s_0 执行动作 $right$ 。这就是 Agent 以 20% 的概率进行探索得到的最优策略，因为增加探索的比例是危险的。如果探索的比例下降，则找到的最优策略可能不同。探索的比例越小，找到最优策略所花费的时间就越长。 ◀

当目标是优化正在探索的 Agent 的值时，SARSA 算法是很有用的。如果你想执行离线学习，并利用不进行探索的 Agent 中的策略，则 Q -学习可能更合适。

476

11.3.7 为路径分配信用和责任

在 Q -学习和 SARSA 算法中，当接收到一个奖赏值时只是修改前一个“状态-行为”对的值。直观的，Agent 通过执行一系列的算法步骤而获得的奖赏，应该和所有的步骤都是有关系的，因此，对于该奖赏，每个步骤均应分配相应的信用和责任。本章介绍其中的一个分配算法。

【例 11-13】 假定存在这样的一个行为“ $right$ ”，它顺序访问状态 s_1 、 s_2 、 s_3 和 s_4 ，且仅当 Agent 从状态 s_3 进入状态 s_4 时才获得一个奖赏；从状态 s_4 开始的任意行为均返回状态 s_1 。同时，存在行为“ $left$ ”，除非在状态 s_4 ，否则它向左走。在 Q -学习和 SARSA 中，当遍历了状态 s_1 、 s_2 、 s_3 和 s_4 ，并接收到奖赏之后，只有 $Q[s_3, right]$ 得到了更新。如果再次访问相同的状态序列，则当进入状态 s_3 时， $Q[s_2, right]$ 将得到更新。 $Q[s_1, right]$ 的更新仅发生在下次从状态 s_1 进入 s_2 的时候。从这个意义上我们说， Q -学习只做了一步备份。

考虑基于因进入状态 s_4 而获得奖赏的 $Q[s_3, right]$ 的更新。站在状态 s_4 的角度看，算法做了一步备份；而站在状态 s_3 的角度看，算法做了超前一步的预测。为了使算法允许将更多的步骤与责任进行挂钩，接收到因进入状态 s_4 而获得的奖赏之后，可以进行两步备份以更新 s_2 的值；换句话说，站在状态 s_2 的角度看，算法做了超前两步的预测以更新 s_2 的值。下面的算法描述基于上述“超前”的概念，但利用的是备份的实现方法。 ◀

假定 Agent 在状态 s_t 执行行为 a_t 后进入状态 s_{t+1} 并接收到奖赏值 r_{t+1} ，接着执行行为 a_{t+1} 后进入状态 s_{t+2} 并接收到奖赏值 r_{t+2} ，在时间 t 超前两步的预测返回值是 $R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+2})$ ，因而 TD 误差为：

$$\delta_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+2}) - Q[s_t, a_t]$$

其中 $V(s_{t+2})$ 是在状态 s_{t+2} 的估计值。两步更新如下：

$$Q[s_t, a_t] \leftarrow Q[s_t, a_t] + \alpha \delta_t$$

不幸的是，这并非是最优 Q 值 Q^* 的一个好的估计，因为行为 a_{t+1} 可能不是最优的。例如，如果执行行为 a_{t+1} 使得 Agent 进入一个奖赏值是 -10 的位置，并且存在更好的行为，则 Agent 不会更新 $Q[s_0, a_0]$ 。然而，多步备份提高了 Agent 实际遵循的策略的估计精度。如果 Agent 遵循的策略记为 π ，则备份给出了一个改进的估计 Q^* 。因此，多步备份可应用于诸如 SARSA 的在策略方法。

假定 Agent 在状态 s_t 执行行为 a_t 后接收到奖赏值 r_{t+1} 并进入状态 s_{t+1} ，接着执行行为 a_{t+1} 后接收到奖赏值 r_{t+2} 并进入状态 s_{t+2} ，以此类推。在时间 t 超前 n ($n \geq 1$) 步的预测返回值记为 $R_t^{(n)}$ ，则

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n})$$

可利用该式和 TD 误差 $R_t^{(n)} - Q[s_t, a_t]$ 更新 $Q[s_t, a_t]$ 。但是，确定 n 的值是不容易的。除了选择一个特定的 n 并向前看 n 步，选择一系列 n 步返回值的平均值是可能的。一种方法是取所有 n 步返回值的加权平均值，其中未来的返回值以指数级衰减(衰减系数为 λ)。这就是所谓的 SARSA(λ) 方法的直观思想。当接收到一个奖赏，更新所有已访问状态的值；对于该奖赏，更早的状态接收更少的信用或责任。

令

$$R_t^i = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$$

其中 $(1-\lambda)$ 是一个归一化常量，用以确保获得一个平均值。下表给出了上式计算结果的细节信息：

向前观察的步数	权重	返回值
1 步	$1-\lambda$	$r_{t+1} + \gamma V(s_{t+1})$
2 步	$(1-\lambda)\lambda$	$r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+2})$
3 步	$(1-\lambda)\lambda^2$	$r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V(s_{t+3})$
4 步	$(1-\lambda)\lambda^3$	$r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \gamma^4 V(s_{t+4})$
...
n 步	$(1-\lambda)\lambda^{n-1}$	$r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^n V(s_{t+n})$
...
总数	1	

合并 r_{t+i} 的同类项，得

$$R_t^i = r_{t+1} + \gamma V(s_{t+1}) - \lambda \gamma V(s_{t+1}) + \lambda \gamma r_{t+2} + \lambda \gamma^2 V(s_{t+2}) - \lambda^2 \gamma^2 V(s_{t+2}) + \lambda^2 \gamma^2 r_{t+3} + \lambda^2 \gamma^3 V(s_{t+3}) - \lambda^3 \gamma^3 V(s_{t+3}) + \lambda^3 \gamma^3 r_{t+4} + \lambda^3 \gamma^4 V(s_{t+4}) - \lambda^4 \gamma^4 V(s_{t+4}) + \cdots$$

一种 SARSA 的变形方法(该方法中的 $V(s_{t+i})$ 的未来估计值是 $Q[s_{t+i}, a_{t+i}]$)将会用到上式的计算。TD 误差，即返回值减去状态的估计值是：

$$R_t^i - Q[s_t, a_t] = r_{t+1} + \gamma Q[s_{t+1}, a_{t+1}] - Q[s_t, a_t] + \lambda \gamma (r_{t+2} + \gamma Q[s_{t+2}, a_{t+2}] - Q[s_{t+1}, a_{t+1}]) + \lambda^2 \gamma^2 (r_{t+3} + \gamma Q[s_{t+3}, a_{t+3}] - Q[s_{t+2}, a_{t+2}]) + \lambda^3 \gamma^3 (r_{t+4} + \gamma Q[s_{t+4}, a_{t+4}] - Q[s_{t+3}, a_{t+3}]) + \cdots$$

不是等到计算结束(可能永远也不会发生)，SARSA(λ) 在未来的每个时间里都更新 $Q[s_t, a_t]$ 的值。当接收到奖赏 r_{t+i} ，Agent 可以利用前述公式中的合适的项的和更新 $Q[s_t, a_t]$ 。由于在所有的时间里都要进行更新操作，所以 $r_{t+3} + \gamma Q[s_{t+2}, a_{t+2}] - Q[s_{t+2}, a_{t+2}]$ 可用于更新所有的先前状态。Agent 可以通过维护一个“资格迹”(eligibility trace)来实现这一点，其中的资格迹规定在每一个时间步中对于一个“状态-行为”对应该更新的量是多少。当一个“状态-行

为”对首先被访问,则它的资格设置为 1。在随后的每个时间步里,它的资格是不断减少的,减少系数为 λr 。如果该“状态-行为”对被再次访问,则其资格加 1。

用一个数组 $e[S, A]$ 来表示资格迹,其中 S 和 A 分别是所有状态的集合和所有行为的集合。在执行每一个行为之后,更新每个“状态-行为”对的 Q 值。

SARSA(λ)算法见图 11-14。

虽然该算法规定对于每一个状态 s 和行为 a ,每当接收到一个新的奖赏就更新 $Q[s, a]$,但仅仅更新那些资格大于某个阈值的不仅更高效,而且准确率的损失也很小。

11.3.8 基于模型的方法

在许多增强学习的应用中,每两个行为之间存在大量的空闲计算时间。例如,一个物理机器人在每两个行为之间可能有许多秒的时间空闲。 Q -学习在每个行为中仅执行一次备份操作,没有充分利用空闲的计算时间。

学习 Q 值的一种可选择的方法是利用数据学习一个模型,即 Agent 利用它的经验显式地学习 $P(s' | s, a)$ 和 $R(s, a, s')$ 。对于环境中执行的每一个动作,Agent 可以异步执行多步的值迭代操作,以给出一个更优的 Q 函数估计。

图 11-15 显示了一个一般的基于模型的增强学习器。与其他的增强学习程序一样,它记录 $Q[S, A]$;但它也维护一个“活力”模型,记为 T ; $T[s, a, s']$ 表示 Agent 在状态 s 执行行为 a 并结束于状态 s' 的次数。和 Dirichlet 分布一样,用该计数与先验计数的和计算概率。这个算法假定一个通用的先验概率。数组 $R[s, a, s']$ 维护在状态 s 执行行为 a 并结束于状态 s' 时接收的奖赏。

在执行完每一个动作之后,Agent 观察奖赏 r 和结果状态 s' ;接着更新转移-计数矩阵 T 和平均奖赏值 R ;然后利用从 T 推导而来的更新概率模型和更新奖赏模型异步执行多步的值迭代操作。该算法存在 3 个未定义的问题:

- 应该更新哪个 Q 值?直观地,算法应该至少更新 $Q[s, a]$,因为接收到了更多的基于转移概率和奖赏的数据。基于此,它可以进行随机更新,也可以确定哪个 Q 值将改变最多。值改变最多的潜在元素是这样的 $Q[s_1, a_1]$,它最有可能在某个已经改变最多的 Q 值(如 $Q[s_1, a_2]$ 已经改变最多)处终止。这可以通过维护一个所考虑的 Q 值的优先队列来加以实现。
- 在两个行为之间应该执行多少步异步值迭代操作? Agent 应该持续地进行异步值迭代操作,直到必须做出决策或获得新的信息。图 11-15 假定 Agent 执行某动作之后等待一个新观察的到来。当来了一个新观察,Agent 就尽快地执行动作。存在该算法的一些变形,例如 Agent 在观察和行动的同时执行循环。这样的 Agent,在它需要的时候执行动作;当它观察到新信息的时候更新转移和奖赏模型。

```

controller SARSA( $\lambda, S, A, \gamma, \alpha$ )
Inputs:
   $S$  是状态的集合
   $A$  是行为的集合
   $\gamma$  是折扣率
   $\alpha$  是步长
   $\lambda$  衰减率
Internal State:
  实值数组  $Q[S, A]$ 
  实值数组  $e[S, A]$ 
  前一状态  $s$ 
  前一行为  $a$ 
begin
  随机初始化  $Q[S, A]$ 
  对于所有的  $s, a$ , 初始化  $e[s, a] = 0$ 
  观察目前的状态  $s$ 
  利用一个基于  $Q$  的策略选择  $a$ 
  repeat forever:
    执行行为  $a$ 
    观察奖赏  $r$  和状态  $s'$ 
    利用一个基于  $Q$  的策略选择  $a'$ 
     $\delta \leftarrow r + \gamma Q[s', a'] - Q[s, a]$ 
     $e[s, a] \leftarrow e[s, a] + 1$ 
    for all  $s'', a''$ 
       $Q[s'', a''] \leftarrow Q[s'', a''] + \alpha \delta e[s'', a'']$ 
     $e[s'', a''] \leftarrow \gamma \lambda e[s'', a'']$ 
     $s \leftarrow s'$ 
     $a \leftarrow a'$ 
  end-repeat
end
  
```

图 11-14 SARSA(λ)

```

controller ModelBasedReinforcementLearner( $S, A, \gamma, c$ )
Inputs:
     $S$  是状态的集合
     $A$  是行为的集合
     $\gamma$  是折扣率
     $c$  是先验计数
Internal State:
    实值数组  $Q[S, A]$ 
    实值数组  $R[S, A, S]$ 
    整数数组  $T[S, A, S]$ 
    状态  $s, s'$ 
    行为  $a$ 
    随机初始化  $Q[S, A]$ 
    随机初始化  $R[S, A, S]$ 
    初始化  $T[S, A, S]=0$ 
    观察目前的状态  $s$ 
    选择并执行行为  $a$ 
repeat forever:
    观察奖赏  $r$  和状态  $s'$ 
    选择并执行行为  $a$ 
     $T[s, a, s'] \leftarrow T[s, a, s'] + 1$ 
     $R[s, a, s'] \leftarrow R[s, a, s'] + \frac{r - R[s, a, s']}{T[s, a, s']}$ 
     $s \leftarrow s'$ 
    repeat
        选择状态  $s_1$  并执行行为  $a_1$ 
        令  $P = \sum_{s_2} (T[s_1, a_1, s_2] + c)$ 
         $Q[s_1, a_1] \leftarrow \sum_{s_2} \frac{T[s_1, a_1, s_2] + c}{P} (R[s_1, a_1, s_2] + \gamma \max_{a_2} Q[s_2, a_2])$ 
    until 获得了一个新观察
  
```

图 11-15 基于模型的增强学习器

- 怎样设定 $R[S, A, S]$ 和 $Q[S, A]$ 的初值？一旦 Agent 观察到一个特定转移 $\langle s, a, s' \rangle$ 的奖赏值，它就使用针对该转移所接收到的奖赏的平均值。然而，对于转移，当更新 Q 的时候，Agent 需要一些从未出现过的值。在面对不确定性的时候，如果采用“探索”的优化策略，则 Agent 可以使用尽可能大的奖赏值（记为 R_{\max} ）作为 R 的初始值，以便鼓励探索。而在值迭代中，将 Q 初始化为与最终的 Q 值越近越好。

图 11-15 的算法假定所有转移 $\langle s, a, s' \rangle$ 的先验计数是相同的。如果先验知识表明一些转移是不可能的，抑或是更可能的，则先验计数应该是不一样的。

该算法也假定奖赏值依赖于初始状态、行为和最终状态。而且假定只有在已经观察到转移真正发生的时候，才知道一个转移 $\langle s, a, s' \rangle$ 的奖赏值。如果奖赏值只依赖于初始状态和行为，则求取 $R[S, A]$ 的效率更高。如果分别存在行为的开销和进入一个状态的奖赏值，且 Agent 能够单独地观察这种开销和奖赏，则可以将奖赏函数分解为 $C[A]$ 和 $R[S]$ ，从而获得更高的学习效率。

比较基于模型和不依赖于模型的增强学习器是困难的。通常，从需要的经验方面看，基于模型的学习器更为高效（需要的经验更少）。然而，不依赖于模型的学习器所需的计算时间更少。应该依据获得经验的难易程度分别进行比较。

11.3.9 基于特征的增强学习

通常进行显式推理的状态是非常多的。与依据状态进行显式推理不同的选择是依据特征进行显式推理。本节讨论基于 Q 函数的逼近的增强学习，这种逼近利用了状态和行为特征的线性组合。这种情形是最简单的，并且工作良好。然而，这种方法需要进行谨慎的特征选择；设计者需要找出合适的特征以表示 Q 函数。这通常是一个困难的工程问题。

基于线性函数逼近的 SARSA 算法

可以在 SARSA 中用特征的线性函数逼近 Q 函数。该算法利用“在策略”方法 SARSA 的原因是，Agent 从真正遵循的策略中进行奖赏的采样（而非从最优的策略进行采样）。

存在许多获取 Q 函数的基于特征的表示方法。本节利用状态和行为两方面的特征构建线性函数。

假定 F_1, \dots, F_n 是状态和行为的数值特征， $F_i(s, a)$ 是状态 s 和行为 a 的第 i 个特征的值。这些特征通常是二元的（值域为 $\{0, 1\}$ ），但也可以取其他值。它们将被用于表示 Q 函数：

$$Q_{\bar{w}}(s, a) = w_0 + w_1 F_1(s, a) + \dots + w_n F_n(s, a)$$

其中， $\bar{w} = \langle w_0, w_1, \dots, w_n \rangle$ 是权值数组。假定存在一个额外的、取值总为 1 的特征 F_0 ，则与其他权值相比， w_0 不再是一种特殊情况。

【例 11-14】 在例 11-8 所示的网格游戏中，一些可能的特征如下：

- 如果行为 a 最大可能地使 Agent 从状态 s 移向怪物将出现的地方，则 $F_1(s, a) = 1$ ；否则 $F_1(s, a) = 0$ 。
- 如果行为 a 最大可能地使 Agent 撞到墙壁，则 $F_2(s, a) = 1$ ；否则 $F_2(s, a) = 0$ 。
- 如果行为 a 最大可能地使 Agent 朝有奖赏的地方移动，则 $F_3(s, a) = 1$ 。
- 如果 Agent 在状态 s 被损坏，且行为 a 将它移向修理站，则 $F_4(s, a) = 1$ 。
- 如果 Agent 被损坏，且行为 a 最大可能地将它移向怪物将出现的地方， $F_5(s, a) = 1$ ；否则 $F_5(s, a) = 0$ 。即 $F_5(s, a)$ 与 $F_1(s, a)$ 完全一样，但前者仅在 Agent 被损坏的情形下适用。
- 如果 Agent 在状态 s 被损坏，则 $F_6(s, a) = 1$ ；否则 $F_6(s, a) = 0$ 。
- 如果 Agent 在状态 s 未被损坏，则 $F_7(s, a) = 1$ ；否则 $F_7(s, a) = 0$ 。
- 如果 Agent 被损坏，且行为 a 的前方存在奖赏，则 $F_8(s, a) = 1$ 。
- 如果 Agent 未被损坏，且行为 a 的前方存在奖赏，则 $F_9(s, a) = 1$ 。
- 如果在状态 s 的 P_0 位置中存在一个奖赏，则 $F_{10}(s, a)$ 取状态 s 中的 x 值，即从左侧墙壁到存在奖赏的位置 P_0 之间的距离。
- 如果在状态 s 的 P_0 位置中存在一个奖赏，则 $F_{11}(s, a)$ 取 $4 - x$ ，其中 x 是从左侧墙壁到存在奖赏的位置 P_0 之间的水平距离。 $4 - x$ 即是从右侧墙壁到存在奖赏的位置 P_0 之间的距离。
- $F_{12}(s, a)$ 到 $F_{29}(s, a)$ 与 F_{10} 和 F_{11} 类似，不同的地方在于存在奖赏的位置和四个方向至墙壁的距离的不同组合。对于奖赏在位置 P_0 的情形，应该考虑与墙壁在 y 方向上的距离。

一个示例性的线性函数是

$$Q(s, a) = 2.0 - 1.0 * F_1(s, a) - 0.4 * F_2(s, a) - 1.3 * F_3(s, a) - 0.5 * F_4(s, a) - 1.2 * F_5(s, a) - 1.6 * F_6(s, a) + 3.5 * F_7(s, a) + 0.6 * F_8(s, a) + 0.6 * F_9(s, a) - 0.0 * F_{10}(s, a) + 1.0 * F_{11}(s, a) + \dots$$

482

483

这些学得的权值(保留一位小数)就是下述算法的一次运行所得到的结果

SARSA 中经验的形式为 $\langle s, a, r, s', a' \rangle$ (Agent 在状态 s 执行了行为 a , 获得奖赏 r 和到达状态 s' , 然后决定执行行为 a'), 它提供新的估计值 $r + \gamma Q(s', a')$ 以更新 $Q(s, a)$ 。该经验可以用作线性回归(linear regression)的一个数据点。令 $\delta = r + \gamma Q(s', a') - Q(s, a)$, 并利用式(7.2), 则权值 w_i 的更新公式如下:

$$w_i \leftarrow w_i + \gamma \delta F_i(s, a)$$

将该公式包含进 SARSA, 得到了如图 11-16 所示的算法。

可以利用一个 ϵ -贪心函数来选择行为 a : Agent 以概率 ϵ 选择一个随机的行为, 否则选择最大化 $Q_{\pi}(s, a)$ 的行为。

虽然该算法易于实现, 但特征工程(feature engineering)(即选择哪些特征)并非易事。线性函数必须不仅要表达执行的最优行为, 还要表达什么特征状态是有用的信息。

存在该算法的许多变形。可以应用不同的函数逼近, 例如神经网络或在叶节点是线性函数的决策树。一种常见的变形是对每一个行为都构造一个单独的函数, 这等价于用这样的一棵决策树逼近 Q 函数: 它在行为上进行分裂, 然后构造线性函数。除此之外, 也可以在其他特征上作分裂。

可以将线性函数逼近与其他方法(例如 SARSA(λ)、 Q -学习或者基于模型的方法)组合在一起。注意, 其中的一些方法具有不同的收敛保证和不同的性能水平。

【例 11-15】 AIspace 网站上有该算法关于例 11-8 所示游戏的一个开源实现, 使用的特征如例 11-14 所示。尝试单步执行该程序, 尽力理解每一步是如何更新参数的; 然后考察多步执行的情况。考虑利用 11.3.5 节所讨论的评估准则度量学习的性能。设法搞清楚所学参数值的含义。

```

1: controller SARSA-FA( $\bar{F}, \gamma, \eta$ )
2:   Inputs:
3:      $\bar{F} = \langle F_1, \dots, F_n \rangle$ : 特征的集合
4:      $\gamma \in [0, 1]$ : 折扣因子
5:      $\eta > 0$ : 梯度下降的步长
6:   Local
7:     随机初始化权值  $\bar{w} = \langle w_0, w_1, \dots, w_n \rangle$ 
8:     观察到目前的状态  $s$ 
9:     选择行为  $a$ 
10:    repeat
11:      执行行为  $a$ 
12:      观察到奖赏  $r$  和状态  $s'$ 
13:      利用一个基于  $Q_{\pi}$  的策略选择  $a'$ 
14:      令  $\delta = r + \gamma Q_{\pi}(s', a') - Q_{\pi}(s, a)$ 
15:      for  $i = 0$  to  $n$  do
16:         $w_i \leftarrow w_i + \eta \delta F_i(s, a)$ 
17:       $s \leftarrow s'$ 
18:       $a \leftarrow a'$ 
19:    until 结束
  
```

图 11-16 基于线性函数逼近的 SARSA 算法

11.4 本章小结

- 期望最大化是一种学习带有隐藏变量(包括类别为隐藏变量的情形)的模型的迭代方法。
- 可以从完全数据中学习信念网络的概率与结构。概率可由计数导出; 结构可通过在给定数据的条件下搜索最优的模型而学得。
- 样本中的缺失值通常并非是随机缺失的。研究这些值缺失的原因通常是重要的。
- 马尔可夫决策过程是增强学习的一种合适的形式化表达。正如函数 $Q(S, A)$ 所表达的那样, 通常的做法是学习在某个状态执行每个动作的估计值。
- 在增强学习中, Agent 应该在利用已有的知识和探索新知识之间进行折衷。
- 离策略学习(例如 Q -学习)方法学习最优策略的值。在策略学习(例如 SARSA)方法学习 Agent 真正执行的策略(包括探索)的值。
- 基于模型的增强学习区别学习状态转换与奖赏模型, 它源于在给定模型的条件下做什么的决策理论规划。

11.5 参考文献及进一步阅读

Fischer[1987]以及 Cheeseman、Kelly、Self、Stutz、Taylor 和 Freeman[1988]介绍了无监督学习。Duda 等[2001]以及 Langley、Iba 和 Thompson[1992]讨论了贝叶斯分类器。Friedman 和 Goldszmidt[1996a]讨论了如何推广朴素贝叶斯分类器,使其允许更多的适当的独立性假设。

信念网络的综述见 Heckerman[1999]、Darwiche[2009]、Koller 和 Friedman[2009]。利用决策树进行结构学习见 Friedman 和 Goldszmidt[1996b]。

增强学习的介绍见 Sutton 和 Barto[1998]以及 Kaelbling、Littman 和 Moore[1996]。Bertsekas 和 Tsitsiklis[1996]研究了函数逼近及其与增强学习的关系。

11.6 习题

11.1 考虑如图 11-1 所示的无监督数据:

- (a) 当 $k=2$ 时, k -均值算法能找出多少种稳定的将样本指派给类的方法?(提示:尝试在一些不同的初始点运行算法,同时考虑什么样的样本指派是稳定的。)不要将类标签的排列视为不同的指派。
- (b) 当 $k=3$ 时, k -均值算法能找出多少种稳定的将样本指派给类的方法?
- (c) 当 $k=4$ 时, k -均值算法能找出多少种稳定的将样本指派给类的方法?
- (d) 为什么有人建议在这个例子中自然的类别个数为 3? 给出“自然”的类别个数的定义,并利用这些数据验证该定义。

11.2 假定对于不断增大的序列值 k 运行 k 均值算法,且对于每一个 k ,算法都运行多次以搜寻具有全局最小误差的样本指派。是否可能存在这种情况:对于一些 k 而言,误差是稳定的;而后性能却大幅提升(即当 $k=3, 4, 5$ 的时候,误差几乎相同,而当 $k=6$ 的时候,误差则小得多)? 如果可能,给出一个例子;否则解释原因。

11.3 给出一个无监督学习的期望最大化算法(图 11-4),它不存储数组 A ,而是为 M 步重复计算合适的值。每一次迭代只需扫描一遍数据集。(提示:对于数据集中的每一个元组,更新所有相关的 M_i 值。)

11.4 假定一个进行 Q -学习的 Agent,具有固定的步长 α 和折扣率 γ ,处于状态 34、执行动作 7、接收奖赏值 3 和结束于状态 65。请问更新的值是多少? 给出其表达式。(越具体越好。)

11.5 在增强学习中,如果 Agent 总是选择最大化 Q 值的行为,则会发生什么情况? 给出两种促使 Agent 进行探索的方法。

11.6 假定进行 Q -学习的 Agent 具有折扣率 γ 和步长 α ,且正在执行一个 ϵ -贪心探索策略。说明 Q -学习与 2.2.1 节的 Agent 体系结构的一致性如何。

- (a) Q -学习 Agent 的信念状态由哪几部分构成?
- (b) 知觉是什么?
- (c) Q -学习 Agent 的功能函数是什么?
- (d) Q -学习 Agent 的信念状态转移函数是什么?

11.7 对于如图 11-12 所示的随时间变化而变化的累积奖赏值曲线图,仅当平衡正负奖赏值是合理行为的时候,最小和零交叉点才是有意义的统计值。当零点并非是合理行为的恰当定义的时候,应该用什么值替换这些统计值?(提示:考虑仅存在正奖赏或负奖赏的情形。)

11.8 对于例 11-8 所示的游戏,比较不同参数设置的结果。特别地,考虑下述情形:

- (a) α 变化, Q 值初始化为 0.0。
- (b) α 变化, Q 值初始化为 5.0。
- (c) α 取 0.1, Q 值初始化为 0.0。
- (d) α 取 0.1, Q 值初始化为 5.0。

(e) 其他设置。

对于每一种情形，多次运行程序，比较下列值的分布：最小值、零交叉点、含有探索的策略的渐近斜率和不含有探索的策略的渐近斜率。对于最后一个任务，在算法收敛之后，将探索参数设为 100%，再多次运行额外的步骤。

11.9 在 Q-学习中，考虑 4 种不同的步长 α_k 取值(注意，在采用变化的 α_k 的 Q-学习中，每一个“状态-行为”对必须有不同的 k 值)：

(i) $\alpha_k = 1/k$ 。

(ii) $\alpha_k = 10/(9+k)$ 。

(iii) $\alpha_k = 0.1$ 。

(iv) 在最开始的 10 000 步中，令 $\alpha_k = 0.1$ ；接下来的 10 000 步中，令 $\alpha_k = 0.01$ ；再接下来的 10 000 步中，令 $\alpha_k = 0.001$ ，依此类推。

回答下列问题：

(a) 理论上，上述哪些情形将收敛到真实的 Q 值？

(b) 实际运行(即运行合理的迭代步数之后)的结果表明哪些情形收敛到真实的 Q 值？再试试其他领域的问题，看看结果怎样。

(c) 当缓慢改变运行环境的时候，结果将如何？

11.10 假定你的朋友给你这样的例子：其中 SARSA(λ)看似给出了不符合直觉的结果。在该例子中，存在两个状态 A 和 B；除了进入状态 A 有一个奖赏值 10 外，没有其他的奖赏或惩罚；有两种行为 *left* 和 *right*，它们仅在状态 B 有区别。在状态 B 执行 *left* 直接进入状态 A，但执行 *right* 后进入状态 A 的概率很小。具体的：

- $P(A|B, \text{left}) = 1$ ；奖赏值是 10。

- $P(A|B, \text{right}) = 0.01$ ；奖赏值是 10。 $P(B|B, \text{right}) = 0.99$ ；奖赏值是 0。

- $P(A|A, \text{left}) = P(A|A, \text{right}) = 0.999$ ， $P(B|A, \text{left}) = P(B|A, \text{right}) = 0.001$ 。这表明进入状态 B 的“资格迹”将与零充分接近。

- γ 和 λ 均取 0.9， α 取 0.4。

假定你的朋友声称在该例子中 $Q(\lambda)$ 并不起作用，原因是在状态 B 执行动作 *right* 的“资格迹”比执行动作 *left* 的大(奖赏和所有参数均取相同的值)。具体地，执行动作 *right* 进入状态 A 的“资格迹”大约为 5，但执行动作 *left* 却为 1。因此，最好的行为是在状态 B 执行动作 *right*，这是不对的。

你的朋友错在哪里？这个例子说明了什么？

11.11 在基于线性函数逼近的 SARSA 算法中，如果运用线性回归最小化 $r + \gamma Q_{\pi}(s', a') - Q_{\pi}(s, a)$ ，将得到与先前不同的结果。说明得到的结果是什么和在正文中描述的结果是否更可取。

11.12 在例 11-14 中，一些特征是完全相关的(例如 F_6 和 F_7)。这些相关的特征是否会影响表达的功能？是否会影响学习的效率？

第四部分

Artificial Intelligence: Foundations of Computational Agents

个体与关系的推理

个体与关系

这是一个具有物理结构的现实世界。通过与这个世界的巨量交互，思维的程序已经得到了训练，因此该程序包含着反映了该世界的结构的代码，并知道如何利用这些代码。这些代码包含了该世界中物理对象的表示，并表达了物理对象间的交互。这些代码多为模块化的……包含了处理不同类型对象的模块及概括了许多类型的对象的模块……模块间以对现实世界做镜像的方式和对世界的演化做出准确预测的方式进行交互……

你利用该世界的结构来做出决策、执行动作。你在哪里画出类别的分界线——类别由单一的对象或单一的对象类组成——由你的思维程序来决定，即你的思维程序做了这个分类工作。这个分类不是随机的，而是反映了世界的一个简洁描述，特别地，是一个有益于利用该世界的结构的描述。

—Eric B. Baum[2004, 169~170 页]

本章讨论的是如何表示个体(事物和对象)及它们之间的关系。正如上面引言中 Baum 建议的那样，现实世界包含了对象，而我们想要的是这些对象的简洁表示。这样的表示可以比仅使用特征的表示方式更为简洁。本章考虑了逻辑表示，并给出了一个详细的例子以说明上述表示如何用做数据库的没有不确定性的自然语言接口。后面的章节介绍本体及符号的意义、关系学习和概率关系模型的意义。

489
}
491

12.1 在特征之外利用结构

人工智能的主要经验之一就是成功的 Agent 利用了世界的结构。前面的几章考虑了使用特征表示的状态。较之明确地表示状态，使用特征来表示状态更为简洁，并且算法能够使用这种简洁性。不过，在这些特征中通常有更多的结构，这些结构可用于表示和推理。特别地，本章考虑了如下方面的推理过程：

- **个体(individual)**：问题域中的事物，它们可以有形的个体，如人和建筑物；虚构的个体，如独角兽和仙女；或是抽象的概念，如课程和时间。
- **关系(relation)**：关于这些个体什么为真。这意味着尽可能地通用，包含了单个体的为真或为假的一元关系，也包括多个个体间的关系。

【例 12-1】 在例 5-5 中，命题 up_{s_2} 、 up_{s_3} 和 ok_{s_2} 没有内部结构。这里没有关于命题 up_{s_2} 和 up_{s_3} 描述的是同一个关系的概念，不过说明了上述关系联系的是不同的个体；也没有说明 up_{s_2} 和 ok_{s_2} 指的是同一个开关。这里没有个体与关系的概念。

一个替代的方法是明确地表示各个开关个体 s_1 、 s_2 、 s_3 和属性(或关系) up 和 ok 。使用这个表示方法，“开关 s_2 是闭合的”表示为 $up(s_2)$ 。知道了 up 和 s_1 的表示后，我们不需要一个单独的 $up(s_1)$ 的定义。可以用一个二元关系，如 $connected_to$ ，来联系两个个体，如 $connected_to(w_1, s_1)$ 。

有多个理由来说明为什么应使用个体和关系而不是仅使用特征：

- 它通常是自然的表示。特征通常是个体的属性，而这个内部结构在转换到特征时

就丢失了。

- Agent 可能不得不在不知道有哪些个体或者有多少个体的情况下在领域中建模, 因此, 也无从知道有哪些特征。在运行期间, Agent 在发现特定环境中的个体时能够构建其特征。
- Agent 能够做一些推理, 而不必关心特定的个体。例如, 它可以获得一些对于所有个体都适用的东西, 而不需知道个体的具体情况。或者, Agent 有可能断定存在一些有特定属性的个体, 而不必关心其他的个体。也可能有一些询问, Agent 能够回答这些询问而不必要区分每一个体。
- 个体的存在可以依赖于动作, 也可以是不确定的。例如, 在制造业中的规划中, 是否有一个工作组件可能依赖于许多其他的子工作组件且这些组件装配正确; 这些组件中的一部分可能依赖于 Agent 的动作, 而另一些可能不在 Agent 的控制之下。因此, Agent 可能不得不在缺少对特征的了解或不知道会有哪些特征的情况下执行动作。
- 通常, Agent 的推理涉及无限多的个体, 因此也涉及无限多的特征。例如, 如果个体是句子, 则 Agent 可能不得不仅推理非常有限的句子集合(例如, 仅包括人们可以说的句子, 或者那些可能产生的合理的句子), 即使可能的句子也是无限的(从而, 特征也是无限的)。

492

12.2 符号与语义

第 5 章是关于符号的推理, 这些符号表示了命题。在本节, 我们扩展语义用于个体和关系的推理。符号可用于表示个体或关系。我们仍然使用命题; 此时, 原子命题有关于关系和个体的内部结构。

图 12-1 说明了带有个体和关系的语义的一般思想。对于符号, 知识库的设计者对其赋予了意义。知识库的设计者知道符号 *kim*、*r123* 和 *in* 在问题域中指代什么, 并且提供了一个用计算机的表示语言编写的句子组成的知识库。这些句子对于设计者是有意义的。他能够使用这些符号提出问题, 而这些符号代表了他所赋予的特定含义。计算机接收到这些句子和问题, 然后计算出答案。计算机不知道这些符号的含义。然而, 为这些符号赋予了含义的人能够使用与这些符号关联的含义在问题世界中解释计算机给出的答案。

493

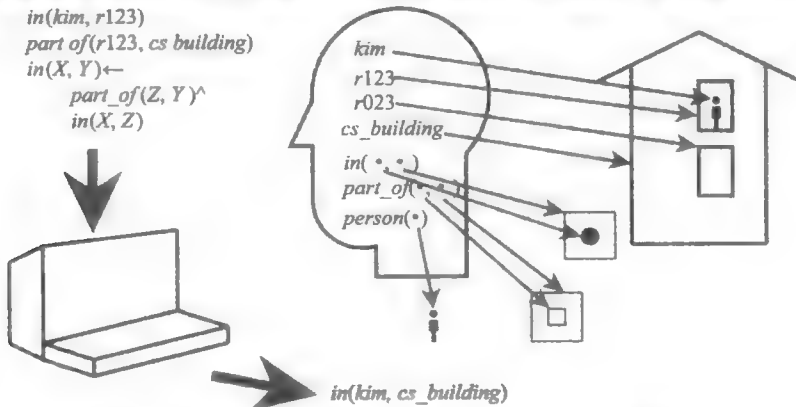


图 12-1 语义的角色。符号的含义在用户的头脑里。计算机处理输入的符号, 并输出符号。输出的符号可由用户根据其赋予这些符号的含义来解释

存在于人脑之中的符号与这些符号所指称的个体和关系的映射称为**概念化**(conceptualization)。在本章,我们假设概念化存在于用户的头脑中,或者是非正式的备注。使概念化变得明确是形式本体的作用。

根据这个观点,正确答案的定义独立于计算出答案的方式。知识库的正确性由语义来定义,而不是通过一个用于证明查询的特定算法来定义。只要推理忠实于语义,可以为提高效率而优化证明程序。这种含义与计算的分离使得 Agent 在优化求解性能的同时仍然保持正确性。

12.3 Datalog: 一个关联规则语言

本节扩展了命题的限定子句语言的语法。对于谓词符号,该语法基于标准的数学符号,不过其中的变量遵循 Prolog 的惯例。

Datalog 的语法(syntax)给定如下,其中字(word)是由字母、数字或下划线("_")组成的序列:

- **逻辑变量**(variable)是以大写字母或下划线开头的字。

例如 *X*、*Room*、*B4*、*Raths* 和 *The_big_guy* 均为变量。

逻辑变量与代数变量或随机变量不同。

- **常量**(constant)是一个以小写字母开头的字,或者是一个数字或字符串。
- **谓词符号**(predicate symbol)是以小写字母开头的字。常量和谓词符号通过它们在知识库中的上下文来识别。

例如, *kim*、*r123*、*f*、*grandfather* 和 *borogroves* 可以是常量也可以是谓词符号,这依赖于它们的上下文; *725* 是常量。

- **项**(term)要么是一个变量,要么是一个常量。

例如 *X*、*kim*、*cs422*、*mome* 或 *Raths* 均可为项。

- 我们扩展**原子符号**(atom symbol)(或简单地称为原子)的定义,其形式如 p 或 $p(t_1, \dots, t_n)$, 其中 p 是一个谓词符号,而每一个 t_i 是一个项。每一个 t_i 是上述谓词的论元(argument)。

例如, *teaches(sue, cs422)*、*in(kim, r123)*、*sunny*、*father(bill, Y)*、*happy(C)* 和 *outgrabe(mome, Raths)* 均可为原子。从原子 *outgrabe(mome, Raths)* 的上下文,我们知道 *outgrabe* 是一个谓词符号,而 *mome* 是一个常量。

这里,限定子句、规则、查询和知识库的概念与命题的限定子句相同,不过原子的定义有所扩展。这些概念的定义重复如下:

- **体**(body)是一个原子或原子的合取。
- **限定子句**(definite clause)是一个原子(称为**原子子句**(atomic clause)),或者具有 $a \leftarrow b$ 的形式(称为**规则**(rule)),其中 a 称为头,是一个原子,而 b 是一个体。我们用一个句点表示子句的结束。
- **知识库**(knowledge base)是一个限定子句的集合。
- **查询**(query)的形式如 *ask b*, 其中 b 是一个体。
- **表达式**(expression)是一个项、原子、限定子句或是一个查询。

在我们的例子中,注释将遵循 Prolog 的惯例,即从一个“%”开始到一行的结束均为注释,它们会被系统忽略。

与传统程序设计语言的关系

本章介绍的逻辑语义的概念应该与传统的程序设计语言(如 Fortran、C++、Lisp 或 Java)中的过程化语义做对比。上述程序设计语言的语义明确了语言结构的含义,而计算机将根据规定的程序执行计算。这一点与这里介绍的证明理论更为接近。逻辑语义为确定符号与世界的关系提供一种途径,也提供了一种程序的结果与如何计算出该结果相互独立的途径。

语义和推理理论的定义对应于 Tarskian 语义和数学逻辑(logic)中的证明的概念。逻辑允许我们定义知识,而不管知识是如何使用的。只要知识库的设计者或用户知道其中知识的含义,就可以验证知识的正确性。人们可以辩论用语言写出的句子是否为真,并通过观察世界来验证其陈述。相同的语义可以用来确定实现的正确性。

个体的概念与在面向对象语言(object oriented language), (如 Smalltalk、C++或Java)中对象的定义类似。主要的区别是面向对象语言中的对象是计算的对象,而不是真实的物理对象。在面向对象语言中,一个“person”对象是一个人(person)的表示,它不是一个真实的人。然而,在 AI 中讨论的表示和推理系统中,一个个体“person”可以表示一个真实的人。在面向对象语言中,对象间相互发送消息。从逻辑方面看,我们不仅想与对象交互,我们还想对它们做出推理。我们可能希望能够预测一个对象将会做什么,而不是让对象做一件事然后观察它做了什么。我们可能希望根据观察到的行为来预测对象的内部状态,例如在诊断病人时的情况。我们甚至想推理、预测个体的行为,而该个体可能试图故意隐藏信息,并且不想让我们知道它做了什么。例如,考虑一个“person”对象:尽管可以与该人进行一些交互,但我们仍然常常对许多信息一无所知。因为我们不能一直询问他来获得这些信息(他可能也不知道或者不想告诉我们),所以我们需要关于该个体的信息的一些外部表示。与一个椅子或一种疾病的交互更加困难,但我们仍然希望对它们做些推理。

许多程序设计语言都有一些处理设计的对象的设施,甚至可能有一个单一的目标。例如,在 Java 中,对象不得不适应单一的类继承机制,而现实中的个体可能有多种角色,属于多种类别;正是这些类间的复杂的交互确定了它们的行为。知识库的设计者可能事先不知道这些类将会如何交互。

【例 12-2】 下面是一个知识库:

```
in(kim,R) ← teaches(kim,cs422) ∧ in(cs422,R)
grandfather(sam,X) ← father(sam,Y) ∧ parent(Y,X)
slithy(toves) ← mimsy ∧ borogroves ∧ outgrabe(mome,Raths)
```

根据上下文, kim、cs422、sam、toves 和 mome 都是常量; in、teaches、grandfather、father、parent、slithy、mimsy、borogroves 和 outgrabe 都是谓词符号;而 X、Y 和 Raths 都是变量。

前两个关于 Kim 和 Sam 的子句可能有一些直观的意义,即使我们还没有为限定子句语言中的句子的含义明确提供任何形式上的说明。然而,如果不考虑助记符提供的暗示信息,对计算机而言,前两个子句并不比第三个有更多的意义。含义仅仅通过语义来表达。 ◀

如果一个表达式不含有任何变量,则它是基(ground)的。例如, teaches(fred, cs322)是一个基表达式,但 teaches(Prof, Couse)不是一个基表达式。

在下一节定义了语义。我们首先考虑基表达式,然后扩展语义以包含变量。

12.3.1 基 Datalog 的语义

给出 Datalog 的语义的第一步是给出基的情况下(不含变量的情况下)的语义。

解释(interpretation)是一个三元组 $I = \langle D, \phi, \pi \rangle$, 其中

- D 是一个称为域(domain)的非空集合。 D 的元素称为个体(individual)。
- ϕ 是一个为每一个常量指派了一个 D 中的元素的映射。
- π 是一个为每一个 n 元谓词符号指派了一个从 D^n 到 $\{true, false\}$ 的函数的映射。

ϕ 是一个从名字映射到域中的个体的函数。称常量 c 指示(denote)了个体 $\phi(c)$ 。这里, c 是一个符号, 而 $\phi(c)$ 可以是任何事物: 可以是一个真实的物理对象, 如一个人或一种病毒; 可以是一个抽象的概念, 如一个课程、爱或者数字 2, 甚至是一个符号。

$\pi(p)$ 指示了 n 元谓词符号 p 对于每一个体的 n 元组是否为真或为假。如果谓词符号 p 没有论元, 那么 $\pi(p)$ 或者为真或者为假。因此, 对于没有论元的谓词符号, 其语义就退化为命题的限定子句的语义。

【例 12-3】 考虑由下面三种位于桌子上的物体构成的世界:



以上述的方式来画出物体是因为它们是世界中的物体, 而不是符号。 \times 是一把剪刀, ☎ 是一部电话, ✎ 是一支铅笔。

假设在我们的语言中, 它们对应的常量为 *phone*、*pencil* 和 *telephone*。我们有谓词 *noisy* 和 *left_of*。假设谓词 *noisy* 是一个一元谓词(它有一个论元), *left_of* 是一个二元谓词(它有两个论元)。

一个表示了桌子上的物体的解释是:

- $D = \{\text{✂}, \text{☎}, \text{✎}\}$
- $\phi(\text{phone}) = \text{☎}, \phi(\text{pencil}) = \text{✎}, \phi(\text{telephone}) = \text{☎}$
- $\pi(\text{noisy})$:

$\langle \text{✂} \rangle$	false	$\langle \text{☎} \rangle$	true	$\langle \text{✎} \rangle$	false
----------------------------	-------	----------------------------	------	----------------------------	-------

$\pi(\text{left_of})$:	$\langle \text{✂}, \text{✂} \rangle$	false	$\langle \text{✂}, \text{☎} \rangle$	true	$\langle \text{✂}, \text{✎} \rangle$	true
	$\langle \text{☎}, \text{✂} \rangle$	false	$\langle \text{☎}, \text{☎} \rangle$	false	$\langle \text{☎}, \text{✎} \rangle$	true
	$\langle \text{✎}, \text{✂} \rangle$	false	$\langle \text{✎}, \text{☎} \rangle$	false	$\langle \text{✎}, \text{✎} \rangle$	false

因为 *noisy* 是一个一元谓词, 所以它只处理一个个体, 并针对每一个个体有一个真值。

因为 *left_of* 是一个二元谓词, 它处理一对个体, 并且在该对个体的第一个元素位于第二个元素的左边时取值为真。因此, $\pi(\text{left_of})(\langle \text{✂}, \text{☎} \rangle) = \text{true}$, 因为剪刀在电话的左边; $\pi(\text{left_of})(\langle \text{✎}, \text{✎} \rangle) = \text{false}$, 因为铅笔不在其自身的左边。

请注意 D 是世界中的物体的集合。关系处在该世界中的物体之间, 而不是在这些物体的名字之间。由于 ϕ 指定 *phone* 和 *telephone* 指向同一件物体, 所以在这个解释下, 对于它们的相同陈述均为真。

【例 12-4】 考虑图 12-1 的一个解释。

D 是一个有 4 个元素的集合: Kim、房间 123($r123$)、房间 023($r023$)和 CS 大楼(CS building)。这不仅是 4 个符号的集合, 而且是包括了真实的人、真实的房间和真实的大楼的集合。很难描述这个集合, 幸运的是, 你永远也不必真的描述它们。为记住它们的含义并把该含义传达给另一个人, 知识库的设计者通常要通过指定物理个体或描绘它们(正像在图 12-1 中那样)并通过自然语言来描述它们的方法来描述 D 、 ϕ 和 π 。

其中的常量为 kim 、 $r123$ 、 $r023$ 和 $cs_building$ 。映射 ϕ 使用灰色的、从每一个常量指向图 12-1 世界中的物体的边来定义。

谓词符号为 $person$ 、 in 和 $part_of$ 。这些谓词的含义就是图中的边所传达的意思。

因此，叫 Kim 的人在房间 $r123$ ，同时也在 CS 大楼里，并且它们是关系 in 的唯一实例(该关系为真)。类似的，房间 $r123$ 和房间 $r023$ 是 CS 大楼的一部分，除此之外，在这个解释中不存在别的为真的 $part_of$ 关系。

需要强调的是， D 中的元素是真实的物理个体，而不是这些个体的名字。名字 kim 不在名字 $r123$ 中，相对的， kim 指示的那个人在 $r123$ 指示的那个房间中。

在一个解释中，每一个基项指代了一个个体。常量 c 指代的解释 I 中的个体表示为 $\phi(c)$ 。

在一个解释中，一个基原子的取值要么为真要么为假。在解释 I 中，如果 $\pi(p)(\langle t'_1, \dots, t'_n \rangle) = true$ (其中 t'_i 是项 t_i 所指代的个体)，则原子 $p(t_1, \dots, t_n)$ 为真；否则为假。

【例 12-5】在例 12-4 中的解释下，原子 $in(kim, r123)$ 为真，因为 kim 所指代的人的确在 $r123$ 所指代的房间里。类似的， $person(kim)$ 为真， $part_of(r123, cs_building)$ 也为真。原子 $in(cs_building, r123)$ 和 $person(r123)$ 在上述解释下为假。

12.3.2 解释变量

当一个变量出现在一个子句中时，该子句在一个解释下为真的条件是对变量的所有可能取值该子句均为真。在子句作用域内，称变量是**全称量化**(universally quantified)的。如果变量 X 出现在子句 C 中，则称 C 在解释下为真就意味着 C 为真且与 X 指代的是域中的哪一个体无关。

为形式地定义变量的语义，一个**变量指派**(variable assignment) ρ 是一个从变量集到域 D 的函数。因此，一个变量指派就为每一个变量指定了域中的一个元素。给定 ϕ 和一个变量指派 ρ ，每一个项都指示了域中的一个个体。如果指定的项是一个常量，则其指代的个体由 ϕ 给出。如果指定的项是一个变量，则其指代的个体由 ρ 给出。给定一个解释和一个变量指派，根据前面的定义，每一个原子非真即假。类似的，给定一个解释和一个变量指派，每一个子句均非真即假。

一个子句在一个解释下为真，如果该子句对于所有的变量指派均为真。这称为**全称量化**。而变量则称为在子句作用域内是**全称量化的**。因此，在一个解释中一个子句为假就意味着存在一个变量指派，在该指派下子句为假。变量的作用域是整个子句，这意味着对于一个子句的所有实例，变量指派都是相同的。

【例 12-6】子句

$$part_of(X, Y) \leftarrow in(X, Y)$$

在例 12-4 的解释中为假，因为在 X 指代 Kim 和 Y 指代房间 $r123$ 这一指派下，子句的体为真，而子句的头为假。子句

$$in(X, Y) \leftarrow part_of(Z, Y) \wedge in(X, Z)$$

为真，因为在任何变量指派下，子句的体和头均为真。

逻辑结论的定义见 5.1.2 节：如果基子句体 g 在 KB 的每一个模型下为真，则称 g 为 KB 的**逻辑结论**(logical consequence)，记为 $KB \models g$ 。

【例 12-7】 假设知识库 KB 为

```

in(kim, r123)
part_of(r123, cs_building)
in(X, Y) ←
    part_of(Z, Y) ∧
    in(X, Z)

```

例 12-4 中定义的解释是 KB 的一个模型，因为在该解释下，每一个子句均为真。

$KB \models in(kim, r123)$ ，因为这一点已明确地在知识库中陈述。如果在一个解释下 KB 中的每一个子句均为真，那么 $in(kim, r123)$ 在那个解释下必为真。

$KB \not\models in(kim, r023)$ 。例 12-4 中定义的解释是 KB 的一个模型，在该模型中 $in(kim, r023)$ 为假。

$KB \models part_of(r023, cs_building)$ 。尽管在例 12-4 的解释下 $part_of(r023, cs_building)$ 为真，但在 KB 中存在着另一个模型，在该模型中 $part_of(r023, cs_building)$ 为假。特别地，对于类似于例 12-4 中的解释的解释，其中

$$\pi(part_of)((\phi(r023), \phi(cs_building))) = false$$

是 KB 的一个模型，在其中 $part_of(r023, cs_building)$ 为假。

$KB \models in(kim, cs_building)$ 。如果在解释 I 下 KB 中的子句为真，则 $in(kim, cs_building)$ 一定为真；否则就会存在一个实例使得 KB 中的第三个子句在解释 I 下为假——这与 I 是 KB 的一个模型相矛盾。

499

请注意语义处理出现在子句体(而不是子句头)中的变量的方式(参见例 12-8)。

【例 12-8】 在例 12-7 中，定义 in 的子句中的变量 Y 在子句水平上是全称量化的；因此，对于所有的变量指派，子句均为真。考虑 X 取值 c_1 ， Y 取值 c_2 的情况。子句

```

in(c1, c2) ←
    part_of(Z, c2) ∧
    in(c1, Z)

```

对于 Z 的所有变量指派均为真。如果存在一个 Z 的变量指派 c_3 ，使得 $part_of(Z, c_2) \wedge in(c_1, Z)$ 在一个解释下为真，那么 $in(c_1, c_2)$ 在该解释下必为真。因此，你可以将例 12-7 的最后一个子句读作“对于所有的 X 和所有的 Y ，如果存在一个 Z 使得 $part_of(Z, Y) \wedge in(X, Z)$ 为真，则 $in(X, Y)$ 为真”。

如果我们希望使量化更加明确，可写为 $\forall X p(X)$ ，可读作“对于所有的 X ， $p(X)$ ”，其含义是对于 X 的每一个变量指派 $p(X)$ 为真。对于 X 的某些变量指派 $p(X)$ 为真的情况记做 $\exists X p(X)$ ，读作“存在 X 使得 $p(X)$ ”。 X 被称为存在量化的变量(existentially quantified variable)。

规则 $P(X) \leftarrow Q(X, Y)$ 的含义是

$$\forall X \forall Y (P(X) \leftarrow Q(X, Y))$$

它等价于

$$\forall X (P(X) \leftarrow \exists Y Q(X, Y))$$

因此，仅出现在体中的自由变量在体的作用域中是存在量化的

有的情况下，我们会对于一些奇异的事情谈论子句是否为真，而这些情况是没有意义的。

【例 12-9】 考虑下面的子句

```

in(cs422, love) ←
    part_of(cs422, sky) ∧
    in(sky, love)

```

其中, *cs422* 指代一个课程, *love* 指代一个抽象的概念, 而 *sky* 指代天空。这里, 根据←的真值表, 在上述特定的解释下, 子句没有意义但值为真, 因为在上述的解释里子句的右边部分为假。

只要头是无意义的, 体也会是无意义的, 因此该规则就永远不能用来证明任何无意义的东西。在检查一个子句是否为真时, 你只需考虑子句的体为真的情况。当体为假时子句恒为真——利用这个规则可简化语义, 且不会导致任何问题, 即使子句没有任何意义。

人类的语义学观点

语义的形式描述没有告诉我们语义为什么是有意思的, 也没有告诉我们它如何成为构建智能系统的基础。使用逻辑背后的基本想法是: 当知识库的设计者想要描述一个特定的世界时, 他们可以选择那个世界作为一个**特定的解释**(intended interpretation), 并针对该解释选择符号的准确含义, 然后写出子句, 表示出在那个世界中为真的东西。当系统计算出知识库的一个逻辑结论时, 知识库的设计者或用户能够根据特定的解释来解释这个结论。因为这个特定的解释是一个模型, 且一个逻辑结论在所有的模型中均为真, 所以逻辑结论在这个特定的解释中必为真。

通常, 为讨论的世界设计一个表示及该表示如何适应形式语义的方法为:

第一步: 选择要表示的任务的域或世界。它们可以是现实世界的某些方面(例如, 大学里的课程结构和学生, 或一个特定时间的实验室环境)、某些虚构的世界(如爱丽丝梦游仙境中的世界, 或一个开关故障时的用电环境的状态), 或一个抽象的世界(如数与集合的世界)。在这个世界中, 记域 D 为所有你希望能够引用或推理的个体或事物的集合。另外, 选择要表示的关系。

第二步: 将描述语言中的常量与世界中你想要命名的个体建立联系。对于 D 中的每一个你想要通过名字引用的元素, 均指派语言中的一个常量。例如, 你可能选择名字“*kim*”来指代一个特定的教授, 名字“*cs322*”来指代特定的 AI 入门课程, 名字“*two*”来指代数字 1 后面的数, 名字“*red*”来指代停止信号灯的颜色。每一个名字都指代了世界中对应的个体。

第三步: 对于每一个你想表达的关系都关联语言中的一个谓词符号。每一个 n 元谓词符号指示了一个从 D^n 到 $\{true, false\}$ 的函数, 该函数指明上述关系在 D^n 的一个子集上为真。例如, 有两个论元(一个教师和一个课程)的谓词符号“*teaches*”可能对应于一个二元关系, 该关系在第一个论元指代的个体讲授了第二个论元指代的课程时为真。这些关系不必为二元的。它们可以有任意个数(零个或多个)的论元。例如, “*is_red*”可能是只有一个论元的谓词。

符号与它们的含义的关联构成了一个特定的解释。

第四步: 现在写下在特定的解释中为真的子句。这通常被称为域的公理化(axiomatizing), 其中给出的子句是域中的公理(axiom)。如果符号 *kim* 指代的那个人确实讲授了符号 *cs502* 指代的课程, 则你可以断言子句 *teaches(kim, cs502)* 在给出的特定的解释中为真。

第五步: 现在你可以提出关于该特定的解释的问题并能使用赋予符号的含义来解释结果。

根据这个方法, 知识库的设计者在第四步之前实际上没有告诉计算机任何事情。前三步在设计者的脑子中进行。当然, 设计者应该为那些符号表示建立文档, 使其他人能够理解设计出的知识库, 从而使得他们记住每一个符号的表示, 也能够检查子句是否为真。计算机不必要访问上述的内容。

世界本身并不规定个体是什么。

【例 12-10】 在一个域的某概念化中, *pink* 可能是有一个论元的谓词符号, 且当论元所

指代的个体为粉色时其值为真。在另一个概念化中, *pink* 可能指代一个颜色为粉色的个体, 且可用做有两个论元的谓词 *color* 的第二个论元的值, 其中谓词 *color* 的含义是第一个论元所指代的个体具有第二个论元所指代的颜色。另外, 一些人可能想在某一水平上描述世界, 此时可能不区分红色的深浅浓淡, 因此在描述中并不包括粉色。另一些人可能想从更细节的水平上来描述世界, 此时粉色显得过于笼统, 所以使用了诸如珊瑚色和鲑肉色来描述。◀

认识到符号的准确含义在知识库设计者的头脑里是重要的。有些时候, 这些符号的准确含义甚至没有写下来; 即使写下来了, 也常常是用自然语言的形式来向其他人传递这些含义。当域中的个体是真实的物理对象时, 常常难以给出其准确含义, 除非指明其实体本身。当个体是一个抽象的个体时——例如大学里的一门课程或爱的概念, 则几乎完全不可能写出它们的准确含义。然而, 这并不能阻止系统表示和推理这类的概念。

【例 12-11】 在例 5-5 中仅用命题表示了图 1-8 中的用电环境。使用个体和关系能够使表示更加直观, 因为关于开关如何工作的通用知识能够清楚地与关于一个特定的房子的知识分开。

为表示这个域, 我们首先要确定域中都有哪些个体。在下文中, 我们设每一个开关、每一盏灯、每一个插座均为个体。我们也将两开关间的电线和开关与电灯间的电线表示为个体。有些人可能会说, 事实上有多对电线使用连接器连接起来, 且电流必须服从基尔霍夫定律。另一些人可能会说上面的抽象也是不合适的, 我们应该为电流的流动建模。不过, 适宜于求解处理的问题的抽象就是一个合适水平的抽象。房子里的居民可能不知道电线个体之间在哪里连接, 也不知道电压是多少。因此, 我们假设一个电流模型, 在其中电流从房子外面通过电线到达电灯。这个模型对于确定电灯是否点亮是合适的, 不过它可能不适合处理所有的任务。

下一步, 为每一个我们想要引用的个体指定名字。这在图 1-8 中已经完成。例如, 个体 w_0 是电灯 l_1 与开关 s_2 间的电线。

下一步, 选择要表示的关系。假设有下面的谓词和与它们联系的特定的解释:

- *light*(L) 为真, 如果 L 指代的个体是一盏灯。
- *lit*(L) 为真, 如果电灯 L 开着并正在发光。
- *live*(W) 为真, 如果有电能进入 W ; 也就是说, W 是通电的。
- *up*(S) 为真, 如果开关 S 是闭合的。
- *down*(S) 为真, 如果开关 S 是断开的。
- *ok*(E) 为真, 如果 E 没有故障; E 可以是断路开关, 也可以是电灯。
- *connected_to*(X, Y) 为真, 如果元件 X 与元件 Y 是连接的, 从而电流能从 Y 流到 X 。

在这个阶段, 我们还没有告诉计算机任何事情。它还不知道有哪些谓词, 更不用说这些谓词的含义了。它还不知道存在哪些个体及这些个体的名字。

在知道关于特定房屋的任何事情之前, 可以向系统输入下面的一般规则:

$lit(L) \leftarrow light(L) \wedge live(L) \wedge ok(L)$

递归的规则可让你根据什么与什么相连来说明哪些是通电的:

$live(X) \leftarrow connected_to(X, Y) \wedge live(Y)$

$live(outside)$

对于特定的房屋, 给定一个元件的特定配置和它们的连接关系, 就可告诉计算机下面关于所讨论的世界的事实:

$light(l_1)$

$light(l_2)$

```

down(s1)
up(s2)
connected_to(w0, w1) ← up(s2)
connected_to(w0, w2) ← down(s2)
connected_to(w1, w3) ← up(s1)

```

这些规则和原子子句就是告诉计算机的全部内容。计算机不知道这些符号的含义。然而，它现在可以回答有关该特定房屋的问题。

◀ 503

12.3.3 带变量的查询

查询用来询问某结论是否是知识库的逻辑结论。使用命题查询，用户可以询问 yes-no 的问题。带参数的查询允许系统返回变量的取值——该取值使查询的事物成为知识库的逻辑结论。

通过使用项替换查询中的变量来得到查询的实例(instance)。同一个变量的多次出现需要使用同一个项来替换。给定一个有自由变量的查询，其解答(answer)或者是查询的一个实例(是知识库的一个逻辑结论)或者是“no”(意味着没有查询的实例可从知识库导出)。通过为查询中的变量提供具体的值得以确定查询的实例。确定查询的哪个实例可从知识库导出的过程被称为解答抽取(answer extraction)。

【例 12-12】 考虑图 12-2 中的子句。写下这些子句的人大概为这些符号联系了一些特定的含义，并且这些子句在某个或许是虚构的世界中是真实的。计算机对房间和方向一无所知。它所知道的仅是那些给定的子句；且它能计算逻辑结论。

```

% imm_west(W, E)为真，如果房间 W 紧接在房间 E 的西边
imm_west(r101, r103)
imm_west(r103, r105)
imm_west(r105, r107)
imm_west(r107, r109)
imm_west(r109, r111)
imm_west(r131, r129)
imm_west(r129, r127)
imm_west(r127, r125)

% imm_east(E, W)为真，如果房间 E 紧接在房间 W 的东边
imm_east(E, W) ←
    imm_west(W, E)

% next_door(R1, R2)为真，如果房间 R1 是房间 R2 的隔壁
next_door(E, W) ←
    imm_east(E, W)
next_door(W, E) ←
    imm_west(W, E)

% two_doors_east(E, W)为真，如果房间 E 在房间 W 东边第二个门
two_doors_east(E, W) ←
    imm_east(E, M) ∧
    imm_east(M, W)

% west(W, E)为真，如果房间 W 在房间 E 的西边
west(W, E) ←
    imm_west(W, E)
west(W, E) ←
    imm_west(W, M) ∧
    west(M, E)

```

图 12-2 一个关于房间的知识库

504
}
505

用户可以提出下面的查询：

```
ask imm_west(r105,r107)
```

而解答是 *yes*。用户可以提出以下查询：

```
ask imm_east(r107,r105)
```

解答仍然是 *yes*。用户可以继续提出以下查询：

```
ask imm_west(r205,r207)
```

此时的解答是 *no*。这意味着查询的内容不是一个逻辑结论，而不是说其为假。在知识库中没有足够的信息来确定房间 *r205* 是否紧接在房间 *r207* 的西边。

查询

```
ask next_door(R,r105)
```

有两个解答。其一是 $R=r107$ ，意味着 $next_door(r107, r105)$ 是子句集合的一个逻辑结论。另一个解答是 $R=r103$ 。查询

```
ask west(R,r105)
```

有两个解答： $R=r103$ 和 $R=r101$ 。查询

```
ask west(r105,R)
```

有三个解答： $R=r107$ 、 $R=r109$ 和 $R=r111$ 。查询

```
ask next_door(X,Y)
```

有 16 个解答，包括了

```
X=r103,Y=r101
```

```
X=r105,Y=r103
```

```
X=r101,Y=r103
```

```
.....
```

12.4 证明与替换

5.2.2 节中的自底向上和自顶向下的命题证明过程均可经过扩展用于 Datalog。

子句的实例通过用项替换子句中的变量来得到。一个特定变量的所有出现位置均替换为相同的项。针对变量而扩展的证明过程必须考虑这样一个事实：子句中的自由变量意味着子句的任何实例均为真。同一个证明过程中可能不得使用同一个子句的不同实例。这种为每一个变量赋什么值的过程被称为替换。

一个替换(substitution)是一个有限的 $\{V_1/t_1, \dots, V_n/t_n\}$ 形式的集合，其中每一个 V_i 是一个不同的变量，每一个 t_i 是一个项。集合中的元素 V_i/t_i 是变量 V_i 的一个绑定(binding)。如果在任何一个项 t_i 中均没有变量 V_i 出现，则称该替换为标准形(normal form)替换。

【例 12-13】 例如， $\{X/Y, Z/a\}$ 是一个标准形替换，其中将 X 绑定为 Y ，将 Z 绑定为 a 。而替换 $\{X/Y, Z/X\}$ 不是一个标准形替换，因为变量 X 既出现在了左边的一个绑定的左边又出现在了右边的一个绑定的右边。

替换 $\sigma = \{V_1/t_1, \dots, V_n/t_n\}$ 在表达式 e 上的应用(application)，记为 $e\sigma$ ，是一个与原始的表达式 e 类似的表达式，不同之处在于 e 中出现 V_i 的地方被替换成了相应的 t_i 。表达式 $e\sigma$ 被称为 e 的一个实例。如果 $e\sigma$ 不包含任何变量，则它被称为 e 的基实例(ground instance)。

【例 12-14】 替换的一些应用如下：

```
 $p(a,X), X/c \rightarrow p(a,c)$ 
```

$$p(Y, c)\{Y/a\} = p(a, c)$$

$$p(a, X)\{Y/a, Z/X\} = p(a, X)$$

$$p(X, X, Y, Y, Z)\{X/Z, Y/t\} = p(Z, Z, t, t, Z)$$

替换可应用于子句、原子和项。例如，将替换 $\{X/Y, Z/a\}$ 应用于子句

$$p(X, Y) \leftarrow q(a, Z, X, Y, Z)$$

得到的结果是子句

$$p(Y, Y) \leftarrow q(a, a, Y, Y, a)$$

如果 $e_1\sigma$ 与 $e_2\sigma$ 相同则称替换 σ 为表达式 e_1 和 e_2 的合一算子。也就是说，两个表达式的合一算子是一个替换，当将该替换应用到每一个表达式上时，会得到相同的结果。

【例 12-15】 $\{X/a, Y/b\}$ 是 $t(a, Y, c)$ 和 $t(X, b, c)$ 的合一算子，因为

$$t(a, Y, c)\{X/a, Y/b\} = t(X, b, c)\{X/a, Y/b\} = t(a, b, c)$$

表达式可以有許多合一算子。

【例 12-16】 原子 $p(X, Y)$ 和 $p(Z, Z)$ 有许多合一算子，包括 $\{X/b, Y/b, Z/b\}$ ， $\{X/c, Y/c, Z/c\}$ 和 $\{X/Z, Y/Z\}$ 。第三个替换比前两个更一般化，因为前两个替换中 X 和 Y 的替换均与 Z 的相同，只不过对其取值有了更多的限制。

替换 σ 是表达式 e_1 和 e_2 的最广合一算子(most general unifier, MGU)，如果满足：

- σ 是两个表达式的合一算子；
- 如果存在另一个替换 σ' 是 e_1 和 e_2 的合一算子，那么对于所有的表达式 e ， $e\sigma'$ 一定是 $e\sigma$ 的一个实例。

如果表达式 e_1 和表达式 e_2 仅是变量名字不同，则表达式 e_1 是表达式 e_2 的一个换名(renaming)。此时，它们相互为对方的一个实例。

如果两个表达式有一个合一算子，则它们至少有一个 MGU。将不同的 MGU 应用到一个表达式后得到的表达式相互间均为换名。也就是说，如果 σ 和 σ' 都是表达式 e_1 和 e_2 的 MGU，那么 $e_1\sigma$ 是 $e_1\sigma'$ 的换名。

【例 12-17】 $\{X/Z, Y/Z\}$ 和 $\{Z/X, Y/X\}$ 都是 $p(X, Y)$ 和 $p(Z, Z)$ 的 MGU。应用它们到表达式 $p(X, Y)$ 得到的结果

$$p(X, Y)\{X/Z, Y/Z\} = p(Z, Z)$$

$$p(X, Y)\{Z/X, Y/X\} = p(X, X)$$

彼此互为换名。

12.4.1 带变量的自底向上过程

通过使用子句的基实例，命题的自底向上证明过程能够扩展应用到 Datalog 中。子句的基实例可通过将子句中的变量替换为常量得到。需要的常量包括出现在知识库或查询中的常量。如果在知识库或查询中没有常量，则需要创建一个。

【例 12-18】 设知识库如下：

$$q(a)$$

$$q(b)$$

$$r(a)$$

$$s(W) \leftarrow r(W)$$

$$p(X, Y) \leftarrow q(X) \wedge s(Y)$$

所有的基实例构成的集合如下：

$$q(a)$$

506

507

```

q(b)
r(a)
s(a) ← r(a)
s(b) ← r(b)
p(a,a) ← q(a) ∧ s(a)
p(a,b) ← q(a) ∧ s(b)
p(b,a) ← q(b) ∧ s(a)
p(b,b) ← q(b) ∧ s(b)

```

5.2.2 节命题的自底向上的证明过程能够用来导出基实例 $q(a)$, $q(b)$, $r(a)$, $s(a)$, $p(a, a)$ 和 $p(b, a)$, 它们都是逻辑结论。

【例 12-19】 设知识库如下：

```

p(X,Y)
g ← p(W,W)

```

自底向上的证明过程必须创建一个新的常量符号，设为 c 。此时所有的基实例如下：

```

p(c,c)
g ← p(c,c)

```

命题的自底向上证明过程将导出 $p(c, c)$ 和 g 。

如果有查询 $\text{ask } p(a, d)$ ，基实例集合会做相应改变来反映出这些常量。

应用于知识库的自底向上证明过程是可靠的，因为每一个规则的每一个实例在每一个模型下均为真。这个过程本质上与无变量的情况相同，不过它使用了子句的基实例集合，而这些子句均定义为真。

这个过程对于基原子也是完备的。也就是说，如果基原子是知识库的一个结论，则该结论终会被导出。为证明这一点，与命题的情形一样，我们构建一个特定的通用模型。一个模型必须明确常量的指代。一个 **Herbrand 解释** 是一个解释，其中的域是象征性的且由语言中的所有常量组成。如果其中没有常量则创建一个个体。在 Herbrand 解释中，每一个常量均指示其自身。

考虑一个 Herbrand 解释，其中关系的基实例为真，而这些实例可通过一个合适的选择规则最终由自底向上的过程导出。显然，这个 Herbrand 解释是给定的规则的一个模型。正如无变量的情形，它是一个**最小模型** (minimal model)，因为在所有的模型中它有最少的原子。如果对于基原子 g 有 $KB \models g$ ，则 g 在最小模型中为真，从而可最终被导出。

【例 12-20】 考虑图 12-2 中的子句。自底向上的证明过程可立即导出 *imm_west* 的每一个已作为事实给出的实例。然后你可以添加 *imm_east* 子句：

```

imm_east(r103,r101)
imm_east(r105,r103)
imm_east(r107,r105)
imm_east(r109,r107)
imm_east(r111,r109)
imm_east(r129,r131)
imm_east(r127,r129)
imm_east(r125,r127)

```

下一步，下面的 *next_door* 关系可加入结论集合中，包括：

```

next_door(r101,r103)
next_door(r103,r101)

```


two_door_east 关系可被加入结论集合中, 包括:

two_door_east(r105,r101)

two_door_east(r107,r103)

最后, 随后的 *west* 关系可被加入结论集合中。

12.4.2 带变量的确定性归结

通过允许在导出过程中使用规则实例, 可扩展命题的自顶向下证明过程以处理带变量的情形。

一个一般解答子句 (generalized answer clause) 具有下面的形式:

$yes(t_1, \dots, t_k) \leftarrow a_1 \wedge a_2 \wedge \dots \wedge a_m$

其中 t_1, \dots, t_k 是项, a_1, \dots, a_m 为原子。yes 的使用使得解答抽取 (answer extraction) 成为可能: 确定查询变量的哪些实例是知识库的逻辑结论。

一开始, 查询 q 的一般解答子句是:

$yes(V_1, \dots, V_k) \leftarrow q$

其中 V_1, \dots, V_k 是出现在 q 中的变量。直观的, 上述子句意味着如果对应的查询的实例为真, 则 $yes(V_1, \dots, V_k)$ 为真。

证明过程维护了一个当前的一般解答子句。

在每一个阶段, 算法在一般解答子句的体中选择一个原子 a_i 。然后在知识库中选择一个头为 a_i 的子句。一般解答子句 $yes(t_1, \dots, t_k) \leftarrow a_1 \wedge a_2 \wedge \dots \wedge a_m$ 在 a_i 上使用选中的子句

$a \leftarrow b_1 \wedge \dots \wedge b_p$

(其中, a_i 和 a 有最广合一算子 σ) 的 SLD 归结是解答子句:

$(yes(t_1, \dots, t_k) \leftarrow a_1 \wedge \dots \wedge a_{i-1} \wedge b_1 \wedge \dots \wedge b_p \wedge a_{i+1} \wedge \dots \wedge a_m) \sigma$

其中在选中的子句的体上的 a_i 在解答子句中已被替换, 且 MGU 也应用于整个解答子句。

一个 SLD 推导是一般解答子句的序列 $\gamma_0, \gamma_1, \dots, \gamma_n$, 其中:

- γ_0 是对应于原始查询的解答子句。如果查询为 q 且带有自由变量 V_1, \dots, V_k , 则最初的一般解答子句 γ_0 为:

$yes(V_1, \dots, V_k) \leftarrow q$

- 通过在 γ_{i-1} 的体中选择一个原子 a_i 来得到 γ_i ; 在知识库中选择一个子句 $a \leftarrow b_1 \wedge \dots \wedge b_p$ 的副本, 该子句的头 a 与 a_i 一致; 使用体 $b_1 \wedge \dots \wedge b_p$ 替换 a_i ; 然后将合一算子应用于整个解答子句中。

这一过程与命题的自顶向下证明过程的主要区别在于, 对于带变量的子句, 证明过程必须从知识库复制子句的副本。在副本中重新命名了子句中的变量。这样做不仅是为了消解变量名字间的冲突, 同时也因为一个证明过程可能使用一个子句的不同实例。

- γ_n 是一个解答。也就是说, 它有如下的形式:

$yes(t_1, \dots, t_k) \leftarrow$

当上面的导出发生时, 算法返回解答:

$V_1 = t_1, \dots, V_k = t_k$

请注意该解答是如何得到的; yes 的论元保存了原始查询中的变量实例的轨迹, 该轨迹通向一个成功的证明。

一个非确定性的过程通过图 12-3 所示的发现 SLD 推导的算法回答了查询。这是一个

非确定性的过程，因为所有的能通过合适的选择找到的推导都不会失败。如果所有的选择都失败了，则算法也会失败，此时不会有导出结果。选择使用搜索来实现。该算法假设 $unify(a_i, a)$ 返回一个 a_i 和 a 的 MGU(如果存在)，或者是 \perp (如果两者不合一)。合一的定义在下一节给出。

```

1: non-deterministic procedure FODCDeductionTD(KB, q)
2:   Inputs
3:     KB: 一个限定子句的集合
4:     查询 q: 带有变量  $V_1, \dots, V_k$ , 需要证明的原子集合
5:   Output
6:     替换  $\theta$ , 如果  $KB \models q\theta$ ; 若无此替换则证明失败
7:   Local
8:     G 是一个一般解答子句
9:     设 G 为一个一般解答子句  $yes(V_1, \dots, V_k) \leftarrow q$ 
10:  while G 不是一个解答 do
11:    假设 G 为  $yes(t_1, \dots, t_k) \leftarrow a_1 \wedge a_2 \wedge \dots \wedge a_n$ 
12:    在 G 的体中选择原子  $a_i$ 
13:    在知识库 KB 中选择子句  $a \leftarrow b_1 \wedge \dots \wedge b_p$ 
14:    将子句  $a \leftarrow b_1 \wedge \dots \wedge b_p$  中的变量改名
15:    将合一算子  $unify(a_i, a)$  设为  $\sigma$ 。如果没有合一算子则返回  $\perp$ 
16:    设 G 为解答子句  $(yes(t_1, \dots, t_k) \leftarrow a_1 \wedge \dots \wedge a_{i-1} \wedge b_1 \wedge \dots \wedge b_p \wedge a_{i+1} \wedge \dots \wedge a_n) \sigma$ 
17:  return  $V_1 = t_1, \dots, V_k = t_k$ 。其中 G 为  $yes(t_1, \dots, t_k) \leftarrow$ 

```

图 12-3 自顶向下限定子句证明过程

【例 12-21】 考虑图 12-2 中的知识库和如下查询：

$ask_two_doors_east(R, r107)$

图 12-4 说明了一个成功的导出过程且得到的解答为 $R=r111$ 。

请注意上述导出过程使用了如下规则的两个实例：

$imm_east(E, W) \leftarrow imm_west(W, E)$

其中一个实例是用 $r111$ 替换 E ，另一个实例是用 $r109$ 替换 E 。

有些选择归结的子句相互对立，可能导致一个无法完成的部分推导。

合一

前面的算法假设我们能够找到两个原子的最广合一算子。合一问题的含义为：给定两个原子，确定它们是否是合一的；如果是合一的，则返回它们的一个 MGU。

求解合一问题的算法在图 12-5 中给出。 E 是意味着合一的等价陈述的集合， S 是一个替换的正确形式的等价集合。在这个算法中，如果 x/y 是 S 中的一个

替换，那么通过替换， x 作为变量不会出现在 S 或 E 中的任何地方。在第 20 行， x 和 y 必须有相同的谓词且必须有相同数量的论元；否则，同一问题就无法求解。

```

yes(R) ← two_doors_east(R, r107)
  归结: two_doors_east(E1, W1) ←
    imm_east(E1, M1) ∧ imm_east(M1, W1)
  替换: {E1/R, W1/r107}
yes(R) ← imm_east(R, M1) ∧ imm_east(M1, r107)
  选择最左边的合取项
  归结: imm_east(E2, W2) ← imm_west(W2, E2)
  替换: {E2/R, W2/M1}
yes(R) ← imm_west(M1, R) ∧ imm_east(M1, r107)
  选择最左边的合取项
  归结: imm_west(r109, r111)
  替换: {M1/r109, R/r111}
yes(r111) ← imm_east(r109, r107)
  归结: imm_east(E3, W3) ← imm_west(W3, E3)
  替换: {E3/r109, W3/r107}
yes(r111) ← imm_west(r107, r109)
  归结: imm_west(r107, r109)
  替换: {}
yes(r111) ←

```

图 12-4 查询 $ask_two_doors_east(R, r107)$ 的导出过程

```

1: procedure Unify( $t_1, t_2$ )
2:   Inputs
3:      $t_1, t_2$ : 原子
4:   Output
5:      $t_1$  和  $t_2$  的最广合一算子; 如果不存在则返回  $\perp$ 
6:   Local
7:      $E$ : 等价陈述的集合
8:      $S$ : 替换
9:      $E \leftarrow \{t_1 = t_2\}$ 
10:     $S \leftarrow \{\}$ 
11:    while  $E \neq \{\}$  do
12:      从  $E$  中选择并移除  $x = y$ 
13:      if  $y$  与  $x$  不完全相同 then
14:        if  $x$  是一个变量 then
15:          将  $E$  和  $S$  中的所有  $x$  替换为  $y$ 
16:           $S \leftarrow \{x/y\} \cup S$ 
17:        else if  $y$  是一个变量 then
18:          将  $E$  和  $S$  中的所有  $y$  替换为  $x$ 
19:           $S \leftarrow \{y/x\} \cup S$ 
20:        else if  $x$  是  $f(x_1, \dots, x_n)$  且  $y$  是  $f(y_1, \dots, y_n)$  then
21:           $E \leftarrow E \cup \{x_1 = y_1, \dots, x_n = y_n\}$ 
22:        else
23:          return  $\perp$ 
24:    return  $S$ 

```

图 12-5 Datalog 中求解合一问题的算法

【例 12-22】 假设我们想求解 $p(X, Y, Y)$ 和 $p(a, Z, b)$ 的合一算子。一开始, E 为 $\{p(X, Y, Y) = p(a, Z, b)\}$ 。在 while 循环的第一次运行中, E 变成了 $\{X = a, Y = Z, Y = b\}$ 。假设下一步选中了 $X = a$ 。那么 S 变成了 $\{X/a\}$, E 为 $\{Y = Z, Y = b\}$ 。假设下一步选中 $Y = Z$ 。那么 S 和 E 中的 Y 就替换为 Z 。 S 成为 $\{X/a, Y/Z\}$, 而 E 成为 $\{Z = b\}$ 。最后选中的是 $Z = b$, 并使用 b 替换 Z , S 就成为 $\{X/a, Y/b, Z/b\}$, E 就成为空集合。替换 $\{X/a, Y/b, Z/b\}$ 就是返回的 MGU。 ◀

12.5 函数符号

对于 Datalog 系统中的推理所涉及的每一个体, 均需要一个用常量表示的名字。通常, 使用个体组件而不是为每一个体分配一个单独的常量来识别个体更为简单。

【例 12-23】 在许多问题域中, 你希望能够引用一段时间, 并将其作为一个个体。你可能会说某一课程的上课时间是 11:30am。你并不想为每一个可能的时间分配一个常量。使用几点几分来定义时间会更好。类似的, 你可能想推理提到了特定日期的事实。你也不想为每一个日期设置一个常量。而使用年、月、日来定义日期是一个更容易的方法。 ◀

使用常量来命名每一个个体就意味着知识库仅能表示有限的个体, 且一旦知识库设计完成后, 其中的个体数也就确定了。然而, 存在着许多的情形, 在该情形中你想在一个无限的个体集合中进行推理。

【例 12-24】 假设你想建造一个系统, 该系统接受英语的提问, 并通过访问一个在线的数据库来回答该提问。此时, 每一句子都被认为是一个个体。你不会想为每一个句子指定一个名字, 因为英语句子数量太多, 不可能逐一为它们命名。为单词命名, 然后使用单词在句子中出现的顺序来明确说明一个句子可能更好。这个方法可能更为实用, 因为需要命名的单词数远少于需要命名的句子数, 且每一个单词都有其自然的命名。你可能想使用

单词中的字母或者单词的构成部分来明确说明单词。

【例 12-25】 你可能想在学生列表中进行推理。例如，你可能需要导出一个班级的学生的平均成绩。一个班级学生的列表是一个个体，它有属性，如列表的长度和它的第七个元素等。尽管为每一列表命名是可能的，但这样做却非常不方便。使用列表的元素来描述列表是一个更好的方法。

函数符号允许你间接地描述个体。与使用常量来描述个体不同，一个个体可使用其他个体来描述。

在句法构成上，一个函数符号(function symbol)是一个以小写字母开头的单词。我们扩展项的定义，因此项(term)可以是变量、常量或具有形式 $f(t_1, \dots, t_n)$ 的符号，其中 f 是一个函数符号， t_i 为项。除扩展了项的定义之外，语言没有其他变化。

项仅出现在谓词符号中。你不会写出逻辑蕴含一个项的子句。然而，你可能会写出包含了原子的子句，其中在原子中使用函数符号作为描述个体的一种方式。

为反映出这一新句法，必须改动语义。此时，仅需作出的改动是 ϕ 的定义。我们扩展 ϕ 使其成为从每一个常量到 D 的每一个元素及从每一个 n 元函数符号到从 D^n 映射到 D 的函数的映射。因此， ϕ 确定了每一函数符号所指示的映射。特别的， ϕ 明确了每一个基项所指代的个体。

由带有函数符号的子句组成的知识库能够计算任何可计算的函数。因此，知识库可解释为一个程序，称为逻辑程序(logic program)。

这个语言上的微小扩展具有重要的影响。仅使用一个函数符号和一个常量，即可引出无限的不同的项和无限的不同的原子。这些无限的项可用来描述无限的个体。

【例 12-26】 假设你想定义一天内的时间，正如例 12-23 中的那样。你可以使用函数符号 am ，从而 $am(H, M)$ 表示时间 $H:Ma.m.$ ，其中 H 是 1~12 的整数， M 是 0~59 的整数。例如， $am(10, 38)$ 表示时间是 10:38 a.m.； am 表示一个将一对整数映射为时间的函数。类似的，你可以定义符号 pm 来表示午后的时间。

使用函数符号的唯一方式是编写使用函数符号来定义关系的子句。这里没有定义函数 am 的意图；计算机中的时间概念并不比其在人的头脑中的概念多。

为使用函数符号，你可以编写对函数符号的论元有量化定义的子句。例如，下面定义了关系 $before(T_1, T_2)$ ，如果一天中的时间 T_1 在时间 T_2 之前，则该关系为真：

```

before(am(H1,M1),pm(H2,M2))
before(am(12,M1),am(H2,M2))←
    H2<12
before(am(H1,M1),am(H2,M2))←
    H1<H2 ∧
    H2<12
before(am(H,M1),am(H,M2))←
    M1<M2
before(pm(12,M1),pm(H2,M2))←
    H2<12
before(pm(H1,M1),pm(H2,M2))←
    H1<H2 ∧
    H2<12
before(pm(H,M1),pm(H,M2))←
    M1<M2

```

这个定义复杂化了，因为上午和下午均从 12 点开始，然后是 1 点，因此，举例来说，12:37 a.m. 在 1:12 a.m. 之前。

函数符号用来构建数据结构。

【例 12-27】 树是一个有用的数据结构。你可以使用树为自然语言处理系统建立一个句子的句法表示。我们可以决定一个有标号的树具有 $node(N, LT, RT)$ 的形式或者具有 $leaf(L)$ 的形式。从而， $node$ 是树中具有一个名字、一个左子树、一个右子树节点的函数。函数符号 $leaf$ 表示树中一个节点的函数。

如果 L 是树 T 中的一个叶子节点的标签，则关系 $at_leaf(L, T)$ 为真。它可定义为：

```
at_leaf(L, leaf(L))
at_leaf(L, node(N, LT, RT)) ←
    at_leaf(L, LT)
at_leaf(L, node(N, LT, RT)) ←
    at_leaf(L, RT)
```

这是一个结构的递归程序的例子。这些规则涵盖了表示树的结构的所有情形。

如果 L 是树 T 的内部节点的标签，则关系 $in_tree(L, T)$ 为真，其定义如下：

```
in_tree(L, node(L, LT, RT))
in_tree(L, node(N, LT, RT)) ←
    in_tree(L, LT)
in_tree(L, node(N, LT, RT)) ←
    in_tree(L, RT)
```

515

【例 12-28】 你可以在列表(list)上推理而不需要内置任何列表的概念。一个列表是一个空表或是一个元素且后面是一个列表。你可以设置一个常量来表示空列表。假设你使用常量 nil 来表示一个空列表，那么你可以选择一个函数符号，比如 $cons(Hd, Tl)$ ，并给它如下的解释：它表示一个列表，其第一个元素是 Hd ，剩余的表是 Tl 。包含了元素 a 、 b 、 c 的列表可表示为：

```
cons(a, cons(b, cons(c, nil)))
```

为使用列表，必须编写使用对应列表的谓词。例如，关系 $append(X, Y, Z)$ ——它在 X 、 Y 和 Z 是列表时为真且结果为 Z 中包含的元素为 X 中的元素后面跟着 Z 中的元素——可递归地定义如下：

```
append(nil, L, L)
append(cons(Hd, X), Y, cons(Hd, Z)) ←
    append(X, Y, Z)
```

这里的 $cons$ 和 nil 没有任何特殊性；我们可以使用 foo 和 bar 来表示它们。

带函数符号的证明过程

对于函数符号的情形，带变量的证明过程继续保留。两种证明过程的主要区别在于前者扩展了项的类别以包含函数符号。

函数符号的使用涉及无限多的项。这就意味着在子句上向前推进推理链时，我们不得不保证为选择子句而设置的标准是公平的。

【例 12-29】 为说明公平的重要性，考虑下面作为一个更大程序的一部分的子句：

```
num(0)
num(s(N)) ← num(N)
```

一个不公平的策略会在最初选择第一个子句来进行推理，而对于后续的推理总是选择第二个子句。第二个子句总能用来导出一个新的结论。这个策略从不会选择其他的子句，也因此从不能导出其他子句的结论。

516

一阶逻辑与二阶逻辑

一阶谓词演算(first-order predicate calculus)是扩展了命题演算以包括带有函数符号和逻辑变量的原子的逻辑。所有的逻辑变量必须使用明确的“全称”(∀)和“存在”(∃)量词。一阶谓词演算的语义就像本章的逻辑程序的语义一样，但有更丰富的运算符。

逻辑程序的语言形成了一阶谓词演算的语用子集，而一阶谓词已经得到了发展，因为它对于许多任务都是有用的。一阶谓词演算可被看做是为逻辑程序增加了析取和明确量化的语言。

一阶逻辑之所以是一阶的，是因为它允许对域中的个体给予量化。一阶逻辑既不允许将谓词作为变量也不允许对谓词进行量化。

二阶逻辑(second-order logic)允许对一阶关系和谓词进行量化，其论元为一阶关系。这些是二阶关系。例如，下面的二阶逻辑公式

$$\forall R \text{ symmetric}(R) \leftrightarrow (\forall X \forall Y R(X, Y) \rightarrow R(Y, X))$$

定义了二阶关系 *symmetric*，如果其论元是对称关系则其为真。

二阶逻辑对于许多应用来说是必需的，因为一阶逻辑不能够定义传递闭包。例如，假设你想使 *before* 作为 *next* 关系的传递闭包关系，其中 *next*(*X*, *s*(*X*))为真。假想 *next* 的含义是“下一毫秒”而 *before* 表示“之前”，则自然的一阶定义可能如下：

$$\forall X \forall Y \text{ before}(X, Y) \leftrightarrow (Y = s(X) \vee \text{before}(s(X), Y)) \quad (12.1)$$

这个表达式没有准确地捕获上述定义，因为如下的例子

$$\forall X \forall Y \text{ before}(X, Y) \rightarrow \exists W Y = s(W)$$

在逻辑上并不服从式(12.1)，因为有式(12.1)的非标准模型中的 *Y* 表示了无限。为捕获上述传递闭包，你需要一个描述了 *before* 是满足上述定义的最小谓词的公式。这可用二阶逻辑来描述。

一阶逻辑是**递归可数**(recursively enumerable)的，这就意味着存在一个可靠、完备的证明过程，从而每一个为真的语句均可在图灵机上使用一个可靠的证明过程得到证明。二阶逻辑不是递归可数的，因此不存在一个可在图灵机上实现的可靠、完备的证明过程。

517

这个永远忽略一些子句的问题被称为**饥饿**(starvation)。一个公平的选择标准是任何可用的子句均有平等的被选择机会。自底向上的证明过程只有在公平选择的条件下才是完备的。

自顶向下证明过程与 Datalog 中的相同(见图 12-3)。合一问题变得更加复杂，因为必须递归地深入项的结构中。在合一算法中有一处改动：变量 *X* 与项 *t* 不合一，其中 *t* 中出现了 *X* 且 *t* 不为 *X*。对这一条件的检查被称为**出现检查**(occurs check)。如果没有使用出现检查且允许一个变量与在其中出现了该变量的项相合一，则证明过程就不再可靠，下面这个例子说明了这一问题。

【例 12-30】 考虑仅有一个子句的知识库：

$$lt(X, s(X))$$

假设特定的解释是在整数域中，其中 *lt* 的含义是“小于”而 *s*(*X*)表示比 *X* 大一的那个整数。查询 ask *lt*(*Y*, *Y*)应该失败，因为在我们特定的解释中子句为假(没有哪个数小于

其自身)。然而,如果 X 和 $s(X)$ 能够合一,则该查询将成功。此时,证明过程将是不可靠的,因为其能够导出在公理模型中为假的结论。◀

图 12-5 中的合一算法经过一个修改可用于找出两个带函数符号的 MGU。如果算法选择了一个等价的关系 $x=y$, 其中 x 是一个变量而 y 是一个含有 x 但不为 x 的项,则该算法应返回 \perp 。最后一步是出现检查。有时候会略去出现检查(如 Prolog), 因为这样可使证明过程更加高效,即使这样会使证明过程变得不可靠。

下面的例子说明了带有函数符号的 SLD 归结的详细内容。

【例 12-31】 考虑下面的子句:

```
append(c(A,X),Y,c(A,Z))←
    append(X,Y,Z)
append(nil,Z,Z)
```

现在先不考虑上述子句的含义。正像计算机那样,仅把它看做是一个符号操纵问题。考虑下面的查询:

```
ask append(F,c(L,nil),c(l,c(i,c(s,c(t,nil))))))
```

518

下面是导出过程:

```
yes(F,L)←append(F,c(L,nil),c(l,c(i,c(s,c(t,nil))))))
    归结: append(c(A1,X1),Y1,c(A1,Z1))←append(X1,Y1,Z1)
    替换: {F/c(l,X1),Y1/c(L,nil),A1/l,Z1/c(i,c(s,c(t,nil)))}
yes(c(l,X1),L)←append(X1,c(L,nil),c(i,c(s,c(t,nil))))
    归结: append(c(A2,X2),Y2,c(A2,Z2))←append(X2,Y2,Z2)
    替换: {X1/c(i,X2),Y2/c(L,nil),A2/i,Z2/c(s,c(t,nil))}
yes(c(l,c(i,X2),L)←append(X2,c(L,nil),c(s,c(t,nil)))
    归结: append(c(A3,X3),Y3,c(A3,Z3))←append(X3,Y3,Z3)
    替换: {X2/c(s,X3),Y3/c(L,nil),A3/s,Z3/c(t,nil)}
yes(c(l,c(i,c(s,X3)),L)←append(X3,c(L,nil),c(t,nil))
```

此时,两个子句均可使用。选择第一个子句则有:

```
    归结: append(c(A4,X4),Y4,c(A4,Z4))←append(X4,Y4,Z4)
    替换: {X3/c(t,X4),Y4/c(L,nil),A4/t,Z4/nil}
yes(c(l,c(i,c(s,X3)),L)←append(X4,c(L,nil),nil)
```

此时,没有一个子句的头可与一般解答子句体中的原子合一。证明失败。

选择第二个子句则有:

```
    归结: append(nil,Z5,Z5).
    替换: {Z5/c(t,nil),X3/nil,L/t}
yes(c(l,c(i,c(s,nil))),t)←
```

此时,证明成功,得到的解答是 $F=c(l,c(i,c(s,nil)))$, $L=t$ 。◀

在本章的后面部分,我们使用 Prolog 中的“句法糖衣”表示法来表示列表。其中空表 nil 记为 $[]$ 。带有第一个元素 E 和剩余列表 R 的表(之前记为 $cons(E,R)$)记为 $[E|R]$ 。有一个记法上的简化: $[X|[Y]]$ 记为 $[X,Y]$, 其中 Y 可为一个序列的值。例如, $[a|[]]$ 记为 $[a]$, $[b|[a|[]]]$ 记为 $[b,a]$; $[a|[b|c]]$ 记为 $[a,b|c]$ 。

【例 12-32】 使用上述的列表表示法,上一例子中的 $append$ 可记为:

```
append([A|X],Y,[A|Z])←
    append(X,Y,Z)
append([],Z,Z)
```

查询

`ask append(F,[L],[l,i,s,t])`

有一个解答 $F=[l, i, s]$, $L=t$ 。证明过程与上一例子的证明过程一致。就证明过程而言, 没有任何变化; 只不过是改变了函数符号和常量的名字。

519

12.6 在自然语言处理中的应用

自然语言处理是一个有意思的领域, 但对于发展和评估其中的表示和推理理论却是困难的领域。AI 中的所有问题在这个领域中都存在; 求解“自然语言问题”与求解“AI 问题”同样困难, 因为任何一个领域均可用自然语言来表达。计算语言学(computational linguistics)领域有丰富的技术和知识。本书仅能给出一个概述。

研究自然语言处理的理由至少有如下三个:

- 你希望计算机使用用户的术语与其用户交流; 你不能强迫用户去学习一门新的语言。这对于普通用户和诸如经理与孩子(他们既没有时间也没有意愿去学习一种新的交互技能)等用户来说尤其重要。
- 现在已经有有了一个使用自然语言记录的、能够使用计算机来访问的、存有巨量信息的数据存储。信息持续地以书籍、新闻、商业和政府报告及科技论文的形式产生出来, 它们中相当大的部分可以在线访问。一个需要大量信息的系统必须能够处理自然语言, 以检索在计算机上可获得的大量信息。
- 在自然语言处理中, 人工智能的许多问题都有非常清晰、明确的形式, 因此它是一个实验通用理论的好领域。

自然语言处理的发展为开发自然语言与知识库和自然语言翻译间的接口提供了可能。我们在下一节说明如何编写一个适用于非常窄的领域的自然语言查询应答系统, 在这个系统中, 格式化的语言就够用了, 且在该语言中只存在少量的歧义(如果有的话)。另外一个极端的例子是浅显但范围宽广的系统, 如例 6-16 和例 7-13 中的帮助系统。开发一个既有深度又有广度的可用系统是困难的。

任何一个自然语言理解理论都具有三个重要的方面:

句法 句法描述了语言的形式。通常使用文法来规定。自然语言要比作为描述逻辑和计算机程序的人工语言的形式语言更加复杂。

语义 语义提供了语言的话语或句子的含义。尽管已有通用的语义理论, 但当我们为一个特定领域建造一个自然语言理解系统时, 我们会努力使用最简单的表示形式。例如, 在下面的开发中, 知识库中有一个从字到概念的固定的映射关系, 该映射对于许多领域来说都是不合适的但却简化了开发。

520

语用 语用解释了话语是如何与环境建立联系的。为理解语言, 一个 Agent 应该考虑更多而不是仅考虑句子; 它必须考虑句子的上下文、环境的状态、讲话者和听众的目标、特殊的惯例等。

为理解这些方面的区别, 考虑下面这些可能出现在 AI 教科书开头的話:

- 这是一本关于人工智能的书籍。
- 绿色青蛙睡眠深沉。
- 无色的绿色观念睡眠狂暴。
- 狂暴地睡眠观念绿色无颜色。

第一句出现在如本书的开头是合适的, 它有很好的句法、语义和语用。第二句有很好的句法和语义, 但出现在一本人工智能书籍的开头则显得很怪异; 从而它在该上下文中没

有很好的语用。后面两句由语言学家 Noam Chomsky[1957]给出。第三句有很好的句法，但其语义却无意义。第四句的句法是错误的；它完全无意义，无论是句法上、语义上和语用上均是如此。

在本书中，我们不想全面介绍计算语言学。关于这方面的介绍请参考本章最后列出的参考文献。

12.6.1 在上下文无关文法中使用限定子句

本节说明如何使用限定子句来表示自然语言的句法和语义。

语言由其合法的句子来定义。句子是符号的序列。合法的句子由文法来规定。

我们的第一个对自然语言的近似是上下文无关文法。一个上下文无关文法(context-free grammar)是一组重写规则(rewrite rule)的集合，使用这些规则将非终结(non-terminal)符号转换成一个终结符号和非终结符号的序列。该语言的句子是一个根据重写规则生成的终结(terminal)符号的序列。例如，下面的文法规则：

$sentence \mapsto noun_phrase, verb_phrase$

意味着一个非终结符号 *sentence*(句子)可以是一个 *noun_phrase*(名词短语)后跟一个 *verb_phrase*(动词短语)。符号“ \mapsto ”的意思是“可重写为”。如果自然语言的一个句子被表示为一个字的列表，则该规则表示如果一个字的列表是一个名词短语后跟一个动词短语，那么这个字的列表就是一个句子：

$sentence(S) \leftarrow noun_phrase(N), verb_phrase(V), append(N, V, S)$

为说明“computer”是一个名词，你要写成：

$noun([computer])$

另有一个上下文无关文法规则的简单表示方法称为限定子句文法(definite clause grammar, DCG)。该文法使用了限定子句，不需要明确的 *append*。每一个非终结符号 *s* 变成一个有两个参数的谓词 $s(T_1, T_2)$ ，该谓词的含义是列表 T_2 是列表 T_1 的结尾，且 T_1 中位于 T_2 之前的字形成了范畴 *s* 的字序列。列表 T_1 和 T_2 一起构成由非终结符号规定的那类字的差异列表(difference list)，因为正是这些差异构成了句法范畴。

【例 12-33】在这个表示方式下，如果列表 T_2 是列表 T_1 的结尾，即 T_1 中位于 T_2 之前的所有字形成了一个名词短语，则 $noun_phrase(T_1, T_2)$ 为真。 T_2 为句子的剩余部分。你可认为 T_2 表示了列表中的一个位置，而该位置在位置 T_1 之后。差异列表表示了这两个位置之间的字。

原子符号

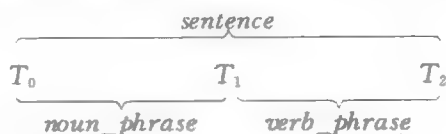
$noun_phrase([the, student, passed, the, course, with, a, computer],$
 $[passed, the, course, with, a, computer])$

在这个特定解释中为真，因为“the student”构成了一个名词短语。

文法规则

$sentence \mapsto noun_phrase, verb_phrase$

的含义是如果在某个 T_0 和 T_1 之间有一个名词短语，且在 T_1 和某个 T_2 之间有一个动词短语，则在该 T_0 与 T_2 之间有一个句子：



这个文法规则可表示为如下的子句：

$sentence(T_0, T_2) \leftarrow$
 $\quad noun_phrase(T_0, T_1) \wedge$
 $\quad verb_phrase(T_1, T_2)$

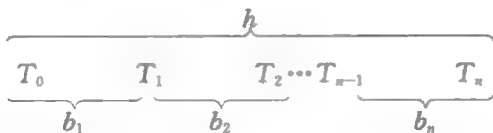
一般的, 规则

522 $h \mapsto b_1, b_2, \dots, b_n$

的含义是 h 由 b_1 后跟 b_2, \dots , 最后跟 b_n 构成, 且可记为下面的限定子句:

$h(T_0, T_n) \leftarrow$
 $\quad b_1(T_0, T_1) \wedge$
 $\quad b_2(T_1, T_2) \wedge$
 $\quad \vdots$
 $\quad b_n(T_{n-1}, T_n)$

并使用下面的解释:



其中 T_i 为新的变量。

如果想说非终结符号 h 映射为终结符号 t_1, \dots, t_n , 则可记为:

$h([t_1, \dots, t_n | T], T)$

并解释为:



因此, 如果 $T_1 = [t_1, \dots, t_n | T_2]$, 则 $h(T_1, T_2)$ 为真。

【例 12-34】 说明非终结符号 h 可重写为非终结符号 a 后跟非终结符号 b , 后跟终结符号 c 和 d , 后跟非终结符号 e , 后跟终结符号 f 和非终结符号 g 的规则可记为:

$h \mapsto a, b, [c, d], e, [f], g$

且可表示为:

$h(T_0, T_6) \leftarrow$
 $\quad a(T_0, T_1) \wedge$
 $\quad b(T_1, [c, d | T_3]) \wedge$
 $\quad e(T_3, [f | T_5]) \wedge$
 $\quad g(T_5, T_6)$

注意, 转换 $T_2 = [c, d | T_3]$ 及 $T_4 = [f | T_5]$ 是人为完成的。

图 12-6 完成了英语的一个简单文法的公理化。图 12-7 给出了字的一个简单字典及其部分语言, 该语言可用图 12-6 中的文法。

523

【例 12-35】 对于图 12-6 中的文法和图 12-7 中的字典, 下面的查询

$ask\ noun_phrase([the, student, passed, the, course, with, a, computer], R)$

将返回

$R = [passed, the, course, with, a, computer]$

句子“The student passed the course with a computer.”有两个不同的解析, 一个是使用子句的如下实例:

$verb_phrase([passed, the, course, with, a, computer], []) \leftarrow$
 $\quad verb([passed, the, course, with, a, computer],$

```

[the,course,with,a,computer]) ∧
noun_phrase([the,course,with,a,computer],[]) ∧
pp([],[])

```

```

% 一个句子是一个名词短语后跟一个动词短语
sentence(T0, T2) ←
    noun_phrase(T0, T1) ∧
    verb_phrase(T1, T2)
% 名词短语是一个限定符后跟修饰符再跟一个名词, 最后跟一个可选的介词短语
noun_phrase(T0, T4) ←
    det(T0, T1) ∧
    modifiers(T1, T2) ∧
    noun(T2, T3) ∧
    pp(T3, T4)
% 修饰符由可能为空的形容词序列组成
modifiers(T, T)
modifiers(T0, T1) ←
    adjective(T0, T1) ∧
    modifiers(T1, T2)
% 可选的介词短语或者为空或者为一个介词后跟一个名词短语
pp(T, T)
pp(T0, T2) ←
    preposition(T0, T1) ∧
    noun_phrase(T1, T2)
% 动词短语是一个动词后跟一个名词短语和一个可选的介词短语
verb_phrase(T0, T1) ←
    verb(T0, T1) ∧
    noun_phrase(T1, T2) ∧
    pp(T2, T3)

```

图 12-6 英语的一个非常有限的子集的一个上下文无关文法

524

另一个使用的实例是:

```

verb_phrase([passed,the,course,with,a,computer],[]) ←
    verb([passed,the,course,with,a,computer],
        [the,course,with,a,computer]) ∧
    noun_phrase([the,course,with,a,computer],[with,a,computer]) ∧
    pp([with,a,computer],[])

```

在第一种情况中, 介词短语修饰的是名词短语(即“the course is with a computer”); 在第二种情况中, 介词短语修饰的是动词短语(即“the course was passed with a computer”)。 ◀

```

det(T, T)
det([a | T], T)
det([the | T], T)
noun([student | T], T)
noun([course | T], T)
noun([computer | T], T)
adjective([practical | T], T)
verb([passed | T], T)
preposition([with | T], T)

```

525

12.6.2 增强文法

图 12-7 一个简单的字典

一个上下文无关文法不足以表达自然语言(如英语)的文法的复杂性。可以添加两种机制使该文法具有更强的表达能力:

- 非终结符号的额外论元。
- 规则中的任意条件。

额外论元将允许我们做如下几件事情: 构建解析树、表示句子的语义结构、增量式地建立表示了针对数据库提出的问题的查询及积累有关短语一致的信息(如数、时态、性和人称上的一致)。

12.6.3 为非终结符号建立结构

你可以为谓词增加一个额外的论元来表示一个解析树、构成形如下面的规则：

```
sentence( $T_0, T_2, s(NP, VP)$ ) ←
    noun_phrase( $T_0, T_1, NP$ ) ∧
    verb_phrase( $T_1, T_2, VP$ )
```

其含义为句子的语法树具有形如 $s(NP, VP)$ 的形式，其中 NP 是针对名词短语的解析树， VP 是针对动词短语的解析树。

如果你想从句法分析中得到一些结果，而不仅是判断句子是否有合法的句法，则上述形式就很重要了。解析树概念是上述需要的简单形式，因为它不能充分表示句子的含义或“深层结构”。例如，你自然想识别出“Alex 讲授 AI 课程”和“AI 课程由 Alex 讲授”具有同一含义，区别仅是分别使用了主动和被动的语气。

12.6.4 封装的文本输出

在文法的定义中没有要求英语输入，也没有要求输出为解析树。一个带有某句子界限意义的文法规则的查询和一个表示了该句子的自由变量可以生成符合上述意义的句子。

文法规则的一个用途是为逻辑项提供封装的文本输出；输出的英语句子与逻辑项相匹配。这对于产生原子、规则和问题的英语版本，使那些可能并不知道这些符号的特定解释，甚至也不知道正规语言的句法的用户能够容易地理解这些生成的内容是很有用的。

526

【例 12-36】 图 12-8 给出了生成关于课表信息的封装文本的文法。例如，下面的查询

```
ask trans(scheduled(w11,cs422,clock(15,30),above(csci333)),T,[])
```

产生的解答为 $T=[the, winter, 2011, session, of, the, advanced, artificial, intelligence, course, is, scheduled, at, 3, :, 30, pm, in, the, room, above, the, computer, science, department, office]$ 。这个列表可写成一个句子提供给用户。

```
% trans(Term,  $T_0, T_1$ ) 为真，如果 Term 转换成的字符串包含在  $T_0$  与  $T_1$  的差异列表中
trans(scheduled( $S, C, L, R$ ),  $T_1, T_2$ ) ←
    trans(session( $S$ ),  $T_1, [of | T_2]$ ) ∧
    trans(course( $C$ ),  $T_2, [is, scheduled, at | T_2]$ ) ∧
    trans(time( $L$ ),  $T_2, [in | T_2]$ ) ∧
    trans(room( $R$ ),  $T_2, T_2$ ).
trans(session(w11), [the, winter, 2011, session |  $T$ ],  $T$ ).
trans(course(cs422), [the, advanced, artificial, intelligence, course |  $T$ ],  $T$ ).
trans(time(clock(0, M)), [12, :, M, am |  $T$ ],  $T$ ).
trans(time(clock(H, M)), [ $H$ , :, M, am |  $T$ ],  $T$ ) ←
     $H > 0 \wedge H < 12$ .
trans(time(clock(12, M)), [12, :, M, pm |  $T$ ],  $T$ ).
trans(time(clock(H, M)), [ $H$ 1, :, M, pm |  $T$ ],  $T$ ) ←
     $H > 12 \wedge$ 
     $H1$  is  $H - 12$ .
trans(room(above( $R$ ))), [the, room, above |  $T_1$ ],  $T_2$  ←
    trans(room( $R$ ),  $T_1, T_2$ )
trans(room(csci333), [the, computer, science, department, office |  $T$ ],  $T$ )
```

图 12-8 封装英语的输出的文法

这个文法可能对于理解自然语言没有用处，因为它需要非常格式化的英语；用户不得不对项进行确切的翻译以得到一个合法的解析。

12.6.5 强约束束

自然语言中施加了一些强约束束，如“a students eat”不是一个合法的句子。句子中的字必须满足一些约定。句子“A students eat”没有满足数上的约定，该约定说明了名词与动词使用单数形式还是复数形式。

527

数的一致可通过参数化非终结符号(引入数并确保语句各部分在数上一致)强制文法中对数的约束。你只需为相关的非终结符号添加一个额外的论元。

【例 12-37】 图 12-9 的文法判定句子“a students”、“the student eat”或“the students eats”非法，因为它们在数上均不一致，但判定句子“a green student eats”、“the students”或“the student”为合法，因为“the”所限定内容可为单数也可复数。

```
% 一个句子是一个名词短语后跟一个动词短语
sentence(T0, T2, Num, s(NP, VP)) ←
    noun_phrase(T0, T1, Num, NP) ∧
    verb_phrase(T1, T2, Num, VP)

% 一个名词短语是空或是一个限定词后跟一个修饰语再跟一个名词及一个可选的介词短语
noun_phrase(T, T, Num, nonp)
noun_phrase(T0, T4, Num, np(Det, Mods, Noun, PP)) ←
    det(T0, T1, Num, Det) ∧
    modifiers(T1, T2, Mods) ∧
    noun(T2, T3, Num, Noun) ∧
    pp(T3, T4, PP)

% 一个动词短语是一个动词后跟一个名词短语再跟一个可选的介词短语
verb_phrase(T0, T3, Num, vp(V, NP, PP)) ←
    verb(T0, T1, Num, V) ∧
    noun_phrase(T1, T2, Num, NP) ∧
    pp(T2, T3, PP)

% 一个可选的介词短语是空或是一个介词后跟一个名词短语。这里仅给出为空的情况。
pp(T, T, nonp)

% 修饰符是一系列的形容词。这里仅给出为空的情况。
modifiers(T, T, [])

% 字典
det([a | T], T, singular, indefinite)
det([the | T], T, Num, definite)
noun([student | T], T, singular, student)
noun([students | T], T, plural, student)
verb([eats | T], T, singular, eat)
verb([eat | T], T, plural, eat)
```

图 12-9 强制数的一致并建立解析树的文法

为解析句子“the student eats”，你可提出查询：

```
ask sentence([the, student, eats], [], Num, T)
```

其返回的解答为：

```
Num=singular,
```

```
T=s(np(definite,[], student, nonp), vp(eat, nonp, nonp))
```

为解析句子“the students eat”，你可提出查询：

```
ask_sentence([the,students,eat],[],Num,T)
```

其返回的解答为:

```
Num=plural,
```

```
T=s(np(definite,[],student,nopp),vp(eat,nonp,nopp))
```

为解析句子“a student eats”，你可提出查询:

```
ask_sentence([a,student,eats],[],Num,T)
```

其返回的解答为:

```
Num=singular,
```

```
T=s(np(indefinite,[],student,nopp),vp(eat,nonp,nopp))
```

请注意，各解答的区别仅在于主语是否为单数及限定词是否是限定的。

12.6.6 建立自然语言与数据库的接口

你可以增强前面的文法以实现一个简单的自然语言与数据库间的接口。这个想法是将短语分解为在数据库中可被查询的形式，而不是转换为解析树的形式。例如，一个名词短语可转换成带有一组谓词的个体，这组谓词用于定义该个体。

【例 12-38】 短语“a female student enrolled in a computer science course”可被转换为:

```
answer(X)←
```

```
female(X) ∧ student(X) ∧ enrolled-in(X,Y) ∧ course(Y)
```

```
∧ department(Y,comp_science)
```

让我们先不考虑量化，如“all”、“a”和“the”是如何转换成量词的。你可以通过名词短语构造一个查询并得到一个个体及由名词短语施加到该个体上的约束列表。适用的文法规则由图 12-10 给定，且该文法规则使用了图 12-11 列出的字典。

```
% 一个名词短语是一个限定词后跟修饰语再跟一个名词及可选的介词短语
noun_phrase(T0, T1, Obj, C0, C1)←
    det(T0, T1, Obj, C0, C1) ∧
    modifiers(T1, T2, Obj, C1, C2) ∧
    noun(T2, T3, Obj, C2, C3) ∧
    pp(T3, T4, Obj, C3, C4)
% 修饰语由一系列的形容词组成
modifiers(T, T, Obj, C, C)
modifiers(T0, T2, Obj, C0, C2)←
    adjective(T0, T1, Obj, C0, C1) ∧
    modifiers(T1, T2, Obj, C1, C2)
% 一个可选的介词短语是空或是一个介词后跟一个名词短语
pp(T, T, Obj, C, C)
pp(T0, T2, O1, C0, C2)←
    preposition(T0, T1, O1, O2, C0, C1) ∧
    noun_phrase(T1, T2, O2, C1, C2)
```

图 12-10 构造一个查询的文法

在这个文法中，

```
noun_phrase(T0, T1, O, C0, C1)
```

的意思是列表 T_1 是列表 T_0 的结尾，且列表 T_0 中位于列表 T_1 之前的字构成一个名词短语。这个名词短语指代了个体 O 。 C_0 是 C_1 的结尾，且出现在 C_1 但不出现在 C_0 的公式是该名词短语施加在个体 O 上的约束。

```

det(T, T, O, C, C).
det([a | T], T, O, C, C).
det([the | T], T, O, C, C).
noun([course | T], T, O, C, [course(O) | C]).
noun([student | T], T, O, C, [student(O) | C]).
noun([john | T], T, john, C, C).
noun([cs312 | T], T, 312, C, C).
adjective([computer, science | T], T, O, C, [dept(O, comp_science) | C]).
adjective([female | T], T, O, C, [female(O) | C]).
preposition([enrolled, in | T], T, O1, O2, C, [enrolled(O1, O2) | C]).

```

图 12-11 用于构造查询的一个字典

【例 12-39】 查询

```
ask noun_phrase([a, computer, science, course], [], Obj, [], C)
```

将返回

```
C=[course(Obj), dept(Obj, comp_science)]
```

查询

```
ask noun_phrase([a, female, student, enrolled, in, a, computer,
science, course], [], P, [], C)
```

返回

```
C=[course(X), dept(X, comp_science), enrolled(P, X), student(P),
female(P)]
```

如果从数据库中查询的是列表 C 中的元素，且上述数据库使用了这些关系和常量，则能够精确地找出选修了计算机科学(computer science)课程的女学生。 ◀

12.6.7 局限

至此为止，我们假设了自然语言的一个非常简单的形式。我们的目标是明确哪些内容能够通过简单的工具来容易地实现，而不是去全面地研究自然语言。有用的数据库前端可用这些工具来构建，而这些工具可通过如有足够约束的域和必要的询问用户(如有歧义的解释时)来表示。

对于自然语言处理的讨论假设了自然语言是可组成的；即整体的含义可由部分的含义导出。组合性在一般情况下是一个不成立的假设。你通常必须知道谈论的上下文和当时的语境以辨别话语的含义。存在许多类型的歧义，这些歧义只能通过理解上下文来解决。 ◀

例如，如果没有上下文和语境的知识，你总是不能够正确确定一个描述的引用。一个描述并不总是指向一个唯一确定的个体。

【例 12-40】 考虑下面的段落：

The student took many courses. Two computer science courses and one mathematics course were particularly difficult. The mathematics course...

段中所指由上下文来定义，而不是仅由描述“The mathematics course”来定义。此处可能有多个数学课程(mathematics course)，但从上下文可知该短语指的是该学生选修的特别难学的那门数学课程。 ◀

如果允许使用“the”或“it”，或者允许字有多种含义，则在数据库应用中就会产生许多

530

531

引用问题。上下文用来消除自然语言中的这些歧义。考虑下面的句子：

Who is the head of the mathematics department?

Who is her secretary?

显然，上述句子中的“her”的指代是清楚的，只要读者理解了“head”是指人，有性的区别，而系(department)却没有性的区分。

12.7 相等

有些时候使用多个项来命名同一个体是有用的。例如，项 4×4 、 2^4 、 $273 - 257$ 及 16 都表示同一个数字。有些时候，你想让每一个名字都引用不同的个体。例如，你希望为大学里的不同课程制定独立的名字。有些时候，你并不知道两个名字是否指代了同一个体，如你不知道早晨 8 点的派送员与下午 1 点的派送员是否是同一个人。

本节讨论相等的作用——它允许我们可否使用两个项来指代环境中的同一个体。请注意，在本章前面介绍的限定子句语言中，所有的解答都是有效的，不管那些项是否指代了同一个体都是如此。

相等是一个特殊的谓词符号，具有一个标准的、独立于域的特定解释。

如果项 t_1 和 t_2 表示了解释 I 中的同一个体，则项 t_1 与项 t_2 在该解释中相等(equal)，记为 $t_1 = t_2$ 。

相等不是相似。如果 a 和 b 均为常量且 $a = b$ ，则不是说有两个事物相似或者是完全一样。与此相反，它的含义是同一事物有两个名字。

【例 12-41】 考虑图 12-12 给出的两把椅子的世界。在这个世界中， $chair1 = chair2$ 不为真，即使这两把椅子完全相同也是如此；如果没有椅子的确切的位置信息，就无法区分它们。此时有一种情况是 $chairOnRight = chair2$ 。此时并不表示位于右边的椅子($chairOnRight$)与 $chair2$ 相似，而是说它就是 $chair2$ 。



图 12-12 两把椅子

12.7.1 允许相等断言

如果你不允许相等断言，那么与一个项相等的就只有它自己。一旦有了相等断言 $X = X$ ，就可以表达相等了。这就意味着对于任一个基项 t ， t 指代了同一个体。

如果你想允许使用相等断言(例如想表达 $chairOnRight = chair2$)，则表示与推理系统必须能够从知识库中导出其相应的结论，其中在数据库中包含某些头部具有相等关系的子句。实现方法主要有两种。第一种类似于谓词那样声明公理化相等。第二种是为相等构建有特殊目的的推理机。本节讨论这两种方法。

1. 公理化相等

相等可以进行如下的公理化。下面三个公理表示相等是自反、对称、传递的。

$$X = X$$

$$X = Y \leftarrow Y = X$$

$$X = Z \leftarrow X = Y \wedge Y = Z$$

其他的公理依赖于语言中函数和关系符号的集合，因此，它们形成了公理模式(axiom schema)。其中的基本思想是你可以使用一个项替换函数和关系中的相等的项。对于每一个 n 元函数符号 f ，有一个如下形式的规则：

$$f(X_1, \dots, X_n) = f(Y_1, \dots, Y_n) \leftarrow X_1 = Y_1 \wedge \dots \wedge X_n = Y_n$$

对于每一个 n 元谓词符号 p , 有一个如下形式的规则:

$$p(X_1, \dots, X_n) \leftarrow p(Y_1, \dots, Y_n) \wedge X_1 = Y_1 \wedge \dots \wedge X_n = Y_n$$

【例 12-42】 二元函数 $cons(X, Y)$ 需要下面的公理:

$$cons(X_1, X_2) = cons(Y_1, Y_2) \leftarrow X_1 = Y_1 \wedge X_2 = Y_2$$

三元关系 $prop(I, P, V)$ 需要下面的公理:

$$prop(I_1, P_1, V_1) \leftarrow prop(I_2, P_2, V_2) \wedge I_1 = I_2 \wedge P_1 = P_2 \wedge V_1 = V_2$$

将这些公理显式地作为知识库的一部分会使得推理效率非常低。在自顶向下深度优先的解释器中使用这些规则不能保证算法能够停止。例如, 对称性公理会造成无限的循环, 除非注意到了相同的子目标。

2. 特殊目的的相等推理

参数化调整(paramodulation)是一个扩充证明过程以实现相等的方法。其基本思想是, 如果有 $t_1 = t_2$, 则任何出现 t_1 的地方均可被 t_2 替代。因此, 相等此时可被看做是**重写规则**(rewrite rule), 在其中用一组项替换另一组相等的项。如果你能够为每一个个体选择一个**规范表示**(canonical representation), 则这个方法工作最好, 其中规范表示是一个项, 而表示该个体的其他的方式可以映射到它。

一个经典的例子是数的表示。存在许多表示了同一个数的项(如 4×4 、 $13 + 3$ 、 $273 - 257$ 、 2^4 、 4^2 、 16), 但我们通常将数字序列(以十进制表示)作为数字的规范表示。

大学使用学号为每一个学生提供了一个规范表示。从而, 同名的不同学生可被区分开来, 同一个学生的不同名字也可映射到该学生的学号上。

12.7.2 唯一名字假设

不可知论者怀疑每一个项的相等, 因此期望着用户以公理化的形式指明哪些名字指代同一个体、哪些名字指代不同个体。与此不同, 采用不同的基项指代不同的个体这样的假设常常会使问题变得更简单。

【例 12-43】 考虑学生数据库的一个示例, 其中每位学生必须有两门科学选修课。假设某位学生已经通过了 $math302$ 和 $psyc303$ 课程; 那么, 如果你知道 $math302 \neq psyc303$, 则你仅仅会知道他们是否已经通过了两门课程。也就是说, 常量 $math302$ 和 $psyc303$ 指代不同的课程。因此, 你必须知道哪些课程号指代不同的课程。与为 n 个个体编写 $n \times (n-1)$ 条不相等公理相比, 假设有每一个课程号指代了不同的课程这一惯例会更好, 并且这样做也可以避免使用不相等公理。

这种处理相等的方法被称为**唯一名字假设**。

唯一名字假设(unique names assumption, UNA)是这样的一个假设: 不同的基项表示不同的个体。也就是说, 对于每一对不同的基项 t_1 和 t_2 , 假设 $t_1 \neq t_2$, 其中“ \neq ”的含义是“不相等”。

请注意, 这并不遵循限定子句语言的语义。对于上述语义来说, 不同的基项 t_1 和 t_2 可以指代同一个体也可指代不同个体。

在目前提出的逻辑中, 唯一名字假设仅仅在子句体中存在明确的不相等或在子句的头中存在明确的相等时才有意义。有了唯一名字假设, 除了前面介绍的在子句中定义相等外, 相等并不出现在子句的头。其他子句意味着相等或者是一类冗余或者是违反了唯一名字公理。

唯一名字假设可使用下面的对不相等的公理模式进行公理化, 其中由对于相等的公理模式连同公理模式共同组成:

- 对于任意不同的常量 c 和 c' , $c \neq c'$ 。
- 对于任意不同的函数符号 f 和 g , $f(X_1, \dots, X_n) \neq g(Y_1, \dots, Y_m)$ 。
- 对于任意的函数符号 f , $f(X_1, \dots, X_n) \neq f(Y_1, \dots, Y_n) \leftarrow X_i \neq Y_i$ 。这里, 对于 n 元函数符号 f , 共有 n 个该模式的实例(对每个 i 都有一项, $1 \leq i \leq n$)。
- 对于任意函数符号 f 和常量 c , $f(X_1, \dots, X_n) \neq c$ 。
- 对于任意出现了 X 的项 t (其中 t 不是项 X), $t \neq X$ 。

有了这个公理化, 两个基项是不相等的当且仅当它们不能合一时, 因为如果它们能够合一则它们必定相同。这个情况对于非基项并不成立。例如, $a \neq X$ 有一些为真的实例(如 X 取值为 b), 而另一些实例为假(如 X 取值为 a)。

唯一名字假设对于数据库应用是非常有用的——你不会希望在数据库中加入诸如 $kim \neq sam$ 、 $kim \neq chris$ 及 $chris \neq sam$ 的子句。唯一名字假设允许我们使用每一个名字指代一个不同的个体这一惯例。

有些时候, 唯一名字假设是不合适的, 如 $2 + 2 \neq 4$ 就是错误的, $clark_kent \neq superman$ 也可能是错误的例子。

有唯一名字假设的自顶向下过程

包含了唯一名字假设的自顶向下过程不应将不相等作为另一个谓词来处理, 这主要是因为对于任意一个给定的个体来说, 存在着太多不同的个体。

如果有一个子目标 $t_1 \neq t_2$, 则对于项 t_1 和 t_2 有如下三种情况:

1) t_1 和 t_2 不能合一。此时, $t_1 \neq t_2$ 是成立的。

例如, $f(X, a, g(X)) \neq f(t(X), X, b)$ 是成立的, 因为这两个项不能合一。

2) t_1 和 t_2 完全一致, 包括了在同一位置使用了同样的变量。此时, $t_1 \neq t_2$ 不成立。

例如, $f(X, a, g(X)) \neq f(X, a, g(X))$ 不成立。

请注意, 对于任何一对基项, 上述两种情况必然出现一个。

3) 另外, $t_1 \neq t_2$ 的一些实例成立, 另一些实例不成立。

例如, 考虑子目标 $f(W, a, g(Z)) \neq f(t(X), X, Y)$ 。函数符号 $f(W, a, g(Z))$ 和 $f(t(X), X, Y)$ 的 MGU 是 $\{X/a, W/t(a), Y/g(Z)\}$ 。一些该不相等的实例——其基实例与合一算子一致——应该不成立。任何与合一算子不一致的实例应该是成立的。与其他的目标不同, 你不会想去枚举每一个成立的实例, 因为那就意味着 X 与每一个函数合一且常量不为 a , 同时为 Y 和 Z 枚举每一对取值且 Y 与 $g(Z)$ 不同。

可扩展自顶向下证明过程以处理唯一名字假设。第一类型的不相等可以成立而第二类不相等不成立。第三类不相等可以推迟(delay), 等待后续目标合一变量, 直到出现前两类不相等中的一个。在图 12-3 中的证明过程中, 在选择 ac 的体中的一个原子时, 为推迟一个目标, 算法应该在不被推迟的原子中选择一个。如果没有其他原子可选且前两种不相等的情况均不出现, 则查询应该成立。总有一个不相等实例成立, 即此实例中的每一个变量取一个不同的值且该取值在其他任何地方均不出现。当发生这种情况时, 用户在解释解答中的自由变量时必须加以注意。此解答不意味着其对于自由变量的每一实例均成立, 而是仅对其中的某些实例成立。

【例 12-44】 考虑判定一位学生是否已通过了至少两门课程的规则:

$passed_two_courses(S) \leftarrow$

```

 $C_1 \neq C_2 \wedge$ 
 $passed(S, C_1) \wedge$ 
 $passed(S, C_2)$ 
 $passed(S, C) \leftarrow$ 
 $grade(S, C, M) \wedge$ 
 $M \geq 50$ 
 $grade(mike, engl101, 87)$ 
 $grade(mike, phys101, 89)$ 
对于查询
 $ask\_passed\_two\_courses(mike)$ 

```

子目标 $C_1 \neq C_2$ 是否成立尚无法确定，因此必须推迟判定。自顶向下证明过程可以选择 $passed(mike, C_1)$ ，其中 C_1 的取值为 *engl101*。随后调用 $passed(mike, C_2)$ ，在其中接着调用 $grade(mike, C_2, M)$ ，此时使用替换 $\{C_2/engl101, M/87\}$ 能够使得推理成立。此时，对于推迟的不相等判断，其变量取值均已绑定且能够做出不相等不成立的结论。

对于 $grade(mike, C_2, M)$ ，另一个可被选择的子句返回的替换为 $\{C_2/phys101, M/89\}$ 。此时，推迟判定的不相等中的变量已经全部绑定，因此足以测试不相等是否成立——这一次不相等是成立的。随后可继续证明 $89 > 50$ ，从而子目标成立。

根据这个例子，人们可能提出这样的问题：“为什么不在最后再判定不相等是否成立？这样不是不需要推迟判定了吗？”对此有两个理由。第一，推迟可能会使效率更高。在这个例子中，对不相等的判定可在测试 $87 > 50$ 是否成立之前。尽管这个特殊的不等式测试很快可以完成，但在许多情况下及早判定是否违反了不相等条件会节约大量的计算时间。第二，如果有一个子证明需要在一个值被绑定之前就被返回，则证明过程需要记住证明过程中的不相等约束，从而，如果将来的合一违反了上述约束则证明过程就会失败。◀

12.8 完备知识假设

为扩展 5.5 节介绍的完备知识假设以用于带有变量和函数符号的逻辑程序中，我们需要关于相等的公理、封闭领域及关于完备性的更复杂的概念。再一次，这里定义了否定即失败(negation as failure)的一种形式。

【例 12-45】 假设一个 *student* 关系定义如下：

```

 $student(mary)$ 
 $student(john)$ 
 $student(ying)$ 

```

根据完备知识假设，可以说上面三个学生即是全部的学生；也就是说，

```

 $student(X) \leftrightarrow X = mary \vee X = john \vee X = ying$ 

```

这就是说，如果 X 是 *mary*、*john* 或者 *ying*，则 X 是一个学生，且如果 X 是一个学生则 X 必为上述三人之一。特别地，Kim 不是一个学生。

要得出 $\neg student(kim)$ 这一结论，需要证明 $kim \neq mary \wedge kim \neq john \wedge kim \neq ying$ 。为导出这些不相等结果，需要使用唯一名字假设。◀

完备知识假设包括唯一名字假设。因此，我们在本节的后续部分假设了关于相等和不相等的公理。

```

子句
 $p(t_1, \dots, t_k) \leftarrow B$ 

```

的 Clark 范式 (clark normal form) 是子句

$$p(V_1, \dots, V_k) \leftarrow \exists W_1 \dots \exists W_m V_1 = t_1 \wedge \dots \wedge V_k = t_k \wedge B$$

其中, V_1, \dots, V_k 是 k 个在原始子句中未出现的变量, W_1, \dots, W_m 是子句中的原始变量。“ \exists ”表示“存在”。当该子句是一个原子子句时, B 为真。

设所有关于 p 的子句均已写为 Clark 范式形式, 且使用同一组引入的变量, 有:

$$p(V_1, \dots, V_k) \leftarrow B_1$$

...

$$p(V_1, \dots, V_k) \leftarrow B_n$$

其等价于:

$$p(V_1, \dots, V_k) \leftarrow B_1 \vee \dots \vee B_n$$

这个蕴含式在逻辑上等价于初始的子句集合。

谓词 p 的 Clark 完备化 (Clark's completion) 是如下的等价:

$$\forall V_1 \dots \forall V_k p(V_1, \dots, V_k) \leftrightarrow B_1 \vee \dots \vee B_n$$

其中, 体中的否定即失败 (\sim) 替换为标准的逻辑否定 (\neg)。完备化意味着 $p(V_1, \dots, V_k)$ 为真当且仅当至少有一个体 B_i 为真。

知识库的 Clark 完备化由每一个谓词符号的完备化以及关于相等和不相等的公理组成。

【例 12-46】 对于子句

student(mary)

student(john)

student(ying)

其 Clark 范式是:

$$student(V) \leftarrow V = mary$$

$$student(V) \leftarrow V = john$$

$$student(V) \leftarrow V = ying$$

其等价于:

$$student(V) \leftarrow V = mary \vee V = john \vee V = ying$$

谓词 *student* 的完备化是:

$$\forall V student(V) \leftrightarrow V = mary \vee V = john \vee V = ying$$

【例 12-47】 考虑如下的递归定义:

passed_each([], *St*, *MinPass*)

passed_each([*C* | *R*], *St*, *MinPass*) \leftarrow

passed(*St*, *C*, *MinPass*) \wedge

passed_each(*R*, *St*, *MinPass*)

用 Clark 范式的形式, 上面的定义可写为:

passed_each(*L*, *S*, *M*) $\leftarrow L = []$

passed_each(*L*, *S*, *M*) \leftarrow

$\exists C \exists R L = [C | R] \wedge$

passed(*S*, *C*, *M*) \wedge

passed_each(*R*, *S*, *M*)

这里, 我们删去了那些明确规定变量更名的相等声明并将变量名改为合适的名字。因此, *passed_each* 的 Clark 完备化为:

$$\forall L \forall S \forall M passed_each(L, S, M) \leftrightarrow L = [] \vee$$

$$\exists C \exists R (L = [C | R] \wedge$$

$passed(S, C, M) \wedge$
 $passed_each(R, S, M)$

在完备知识假设之下，仅使用限定子句无法定义的关系现在可以定义了。

【例 12-48】 假设你有一个数据库，其中当 C 为一个课程时 $course(C)$ 为真，且有 $enrolled(S, C)$ ，其含义为学生 S 在课程 C 的选课名单中。如果没有完备知识假设，则当没有学生在课程 C 的选课名单中时，你不能定义 $empty_course(C)$ 为真。这是因为总有一个知识库模型，在该模型中有某人在所有课程的选课名单中。

使用否定即失败规则， $empty_course(C)$ 可被定义为：

$empty_course(C) \leftarrow course(C) \wedge \neg has_Enrollment(C)$
 $has_Enrollment(C) \leftarrow enrolled(S, C)$

其完备化为：

$\forall C empty_course(C) \leftrightarrow course(C) \wedge \neg has_Enrollment(C)$
 $\forall C has_Enrollment(C) \leftrightarrow \exists S enrolled(S, C)$

这里，我们要提醒一下。当在否定即失败规则中含有自由变量时，你需要非常小心。此时的结果通常与你的设想不一致。在前面的例子中，我们引入了谓词 $has_Enrollment$ 以避免在否定即失败中含有自由变量。下面考虑如果没有做上述处理时会发生什么。

540

【例 12-49】 有人可能试图将 $empty_course$ 定义为如下形式：

$empty_course(C) \leftarrow course(C) \wedge \neg enrolled(S, C)$

其完备化为：

$\forall C empty_course(C) \leftrightarrow \exists S course(C) \wedge \neg enrolled(S, C)$

这是不正确的。设有如下子句：

$course(cs422)$
 $course(cs486)$
 $enrolled(mary, cs422)$
 $enrolled(sally, cs486)$

子句

$empty_course(cs422) \leftarrow course(cs422) \wedge \neg enrolled(sally, cs422)$

是前述子句的一个实例，其体为真，但其头为假，因为 $cs422$ 不是一个没有学生的课程。这是前述子句的正确性的一个反例。

请注意，例 12-48 中定义的完备化等价于：

$\forall C empty_course(C) \leftrightarrow course(C) \wedge \neg \exists S enrolled(S, C)$

存在量词在否定的范围中，所以其等价于：

$\forall C empty_course(C) \leftrightarrow course(C) \wedge \forall S \neg enrolled(S, C)$

完备知识假设证明过程

带有变量和函数的否定即失败的自顶向下证明过程与命题的否定即失败的自顶向下证明过程非常相似。因为有唯一名字假设，则在否定目标中有自由变量时会产生一个问题。

【例 12-50】 考虑下面的子句：

$p(X) \leftarrow \neg q(X) \wedge r(X)$
 $q(a)$
 $q(b)$
 $r(d)$

540

根据语义, 对于查询 $\text{ask } p(X)$ 只有一个解答, 即 $X=d$ 。因为有 $r(d)$ 成立, 所以 $\sim q(d)$ 成立, 进而 $p(d)$ 在逻辑上可从知识库导出。

当自顶向下证明过程遇到 $\sim q(X)$ 时, 它不应该试图去证明 $q(X)$ 为真而 $\sim q(X)$ 为假 (使用替换 $\{X/a\}$)。这将会使目标 $p(X)$ 不成立, 然而该目标应该成立。因此, 该证明过程不是完备的。请注意, 如果知识库包含了 $s(X) \leftarrow \sim q(X)$, 则 $q(X)$ 不成立就意味着 $s(X)$ 成立。因此, 根据否定即失败, 非完备性就会导致不可靠性。

根据唯一名字假设 (见 12.7.2 节), 一个可靠的证明过程应该推迟否定的子目标, 直到自由变量有了绑定的取值。

当含有对带有自由变量的否定即失败的调用时, 我们需要一个更加复杂的自顶向下证明过程:

- 包含了自由变量的否定即失败目标必须推迟到变量变成绑定之时。
- 如果变量永不会变成绑定的, 则出现目标紊乱 (flounder)。此时, 你不能对目标作出任何结论。下面的例子说明你应该对此时紊乱的目标做一些更复杂的事情。

【例 12-51】 考虑子句:

$$p(X) \leftarrow \sim q(X)$$

$$q(X) \leftarrow \sim r(X)$$

$$r(a)$$

及查询

$$\text{ask } p(X)$$

完备化的知识库是:

$$p(X) \leftrightarrow \neg q(X)$$

$$q(X) \leftrightarrow \neg r(X)$$

$$r(X) \leftrightarrow X=a$$

对于 r , 使用替换 $X=a$ 可得到 $q(X) \leftrightarrow \neg X=a$, 因此有 $p(X) \leftrightarrow X=a$ 。从而有一个解答, 即 $X=a$, 不过推迟该目标将不会有助于发现此解答。一个证明过程应该分析未能得出该解答的情况。然而, 这样的过程超出了本书讨论的范围。

12.9 本章小结

- 在使用个体和关系刻画的领域中, 表示个体的常量及表示关系的谓词符号均可用于推理, 以确定在域中哪些为真。
- Datalog 是带有常量、全称量化的变量、关系和规则的逻辑语言。
- 替换用于生成原子和规则的实例。合一使得原子变得相同, 以便用于证明。
- 函数符号用于指代一个可能的无限的个体集合, 该集合中的个体使用其他个体来描述。函数符号可用于构建数据结构。
- 使用限定子句来表示自然语言的文法是可能的。
- 项之间的相等意味着这些项均指代同一个体。
- Clark 完备化可在完备知识假设条件下用于定义否定即失败的语义。

12.10 参考文献及进一步阅读

Datalog 和逻辑程序由 Kowalski[1979]、Sterling 和 Shapiro[1986] 及 Garcia-Molina、Ullman 和 Widom[2009] 描述。逻辑程序设计的历史可参考 Kowalski[1988] 和 Colmerauer、Roussel[1996] 的工作。

关于否定即失败的工作, 以及唯一名字假设的工作, 均以 Clark[1978] 的工作为基础。要大致了解逻

541

辑程序设计和深入了解否定即失败,可参考 Lloyd[1987]的书。Apt 和 Bol[1994]对各种处理否定即失败的技术做了总结。

想了解计算语言学请参考 Jurafsky 和 Martin[2008]的工作及 Manning 和 Schütze[1999]的工作。使用限定子句来描述自然语言由 Pereira 和 Shieber[2002]及 Dahl[1994]完成。

12.11 习题

- 12.1 考虑一个有两个个体(\mathbf{x} 和 \mathbf{y})、两个谓词符号(p 和 q)及三个常量(a 、 b 和 c)的域。知识库 KB 的定义如下:

$$p(X) \leftarrow q(X)$$

$$q(a)$$

- (a) 给出一个解释,该解释是 KB 的一个模型。
 (b) 给出一个解释,该解释不是 KB 的一个模型。
 (c) 共有多少个解释?给出你的理由。
 (d) 上述解释中有多少是 KB 的模型?给出你的理由。

542

- 12.2 考虑包含了常量符号 a 、 b 和 c ,谓词符号 p 和 q ,没有函数符号的语言。我们有下面的知识库,其从该语言中构建。

$$KB_1 = \{p(a)\}$$

$$KB_2 = \{p(X) \leftarrow q(X)\}$$

$$KB_3 = \{p(X) \leftarrow q(X),$$

$$p(a),$$

$$q(b)\}$$

现在考虑该语言的形如 $I = \langle D, \pi, \phi \rangle$ 的可能解释,其中 $D = \{\mathbf{x}, \mathbf{y}, \mathbf{a}, \mathbf{b}\}$ 。

- (a) 在我们的简单语言中,对于域中的这4种元素共有多少种解释?请说出你的理由。(提示:考虑对于常量符号有多少可能的指派 ϕ ,并考虑谓词 p 和 q 有多少种扩展,最后确定存在多少指派 π)不要试图枚举所有可能的解释。
 (b) 在上述解释中,有多少是 KB_1 的模型?请给出你的理由。
 (c) 在上述解释中,有多少是 KB_2 的模型?请给出你的理由。
 (d) 在上述解释中,有多少是 KB_3 的模型?请给出你的理由。
- 12.3 考虑下面的知识库:

$$r(a)$$

$$r(e)$$

$$p(c)$$

$$q(b)$$

$$s(a, b)$$

$$s(d, b)$$

$$s(e, d)$$

$$p(X) \leftarrow q(X) \wedge r(X)$$

$$q(X) \leftarrow s(X, Y) \wedge q(Y)$$

请给出可从该知识库中导出的基原子结论集合。假设使用了自底向上的证明方法且在每一次迭代中都在可选子句中按顺序选择第一个。进一步,可用的常量替换均按“字母顺序”进行选择(如果有多个可选);例如,如果在某次迭代中替换 X/a 和 X/b 均可适用,则首选 $q(a)$ 。那么按顺序导出的结论是怎样的?

- 12.4 在例 12-21 中,算法偶然地选择了 $imm_west(r109, r111)$ 作为归结的子句。如果选择的是另一个子句会产生什么样的结果?请给出归结过程,并给出不同的解答或者给出一个泛化的解答子句,该子句不能够使用知识库中的任意子句进行归结。

543

12.5 在例 12-21 中, 我们总是选择最左边的合取项进行归结。存在一个选择规则(在查询中选择一个合取项进行归结), 对于本例, 该规则将仅返回一个选项吗? 给出一个一般规则, 其产生较少的失败分支(至少对于本例是如此)。给出一个你的规则不能工作的例子。

12.6 按照例 12-21 的方式, 给出下面查询的导出过程:

(a) *ask two_doors_east*(*r107*, *R*)

(b) *ask next_door*(*R*, *r107*)

(c) *ask west*(*R*, *r107*)

(d) *ask west*(*r107*, *R*)

请给出每一个查询的解答。

12.7 考虑下面的知识库:

has_access(*X*, *library*) \leftarrow *student*(*X*)

has_access(*X*, *library*) \leftarrow *faculty*(*X*)

has_access(*X*, *library*) \leftarrow *has_access*(*Y*, *library*) \wedge *parent*(*Y*, *X*)

has_access(*X*, *office*) \leftarrow *has_keys*(*X*)

faculty(*diane*)

faculty(*ming*)

student(*william*)

student(*mary*)

parent(*diane*, *karen*)

parent(*diane*, *robyn*)

parent(*susan*, *sarah*)

parent(*sarah*, *ariel*)

parent(*karen*, *mary*)

parent(*karen*, *todd*)

(a) 给出查询 *ask has_access*(*todd*, *library*) 的 SLD 导出过程。

(b) 查询 *ask has_access*(*mary*, *library*) 有两个 SLD 导出, 请给出它们。

(c) 查询 *ask has_access*(*ariel*, *library*) 有 SLD 导出吗? 请解释。

(d) 请解释查询 *ask has_access*(*X*, *office*) 的解答为什么为空?

(e) 假设下面的子句加入了知识库:

has_keys(*X*) \leftarrow *faculty*(*X*)

那么查询 *ask has_access*(*X*, *office*) 的解答是什么?

12.8 应用了下面的替换后得到的结果是什么?

(a) $f(A, X, Y, X, Y)\{A/X, Z/b, Y/c\}$

(b) $\text{yes}(F, L) \leftarrow \text{append}(F, c(L, \text{nil}), c(L, c(i, c(s, c(t, \text{nil}))))))$
 $\{F/c(L, X_1), Y_1/c(L, \text{nil}), A_1/l, Z_1/c(i, c(s, c(t, \text{nil})))\}$

(c) $\text{append}(c(A_1, X_1), Y_1, c(A_1, Z_1)) \leftarrow \text{append}(X_1, Y_1, Z_1)$
 $\{F/c(L, X_1), Y_1/c(L, \text{nil}), A_1/l, Z_1/c(i, c(s, c(t, \text{nil})))\}$

12.9 找出下面表达式对的最广合一算子:

(a) $p(f(X), g(g(b)))$ 和 $p(Z, g(Y))$

(b) $g(f(X), r(X), t)$ 和 $g(W, r(Q), Q)$

(c) $\text{bar}(\text{val}(X, \text{bb}), Z)$ 和 $\text{bar}(P, P)$

12.10 对于下面的原子对, 如果有最广合一算子请给出, 否则请解释为什么不存在:

(a) $p(X, Y, a, b, W)$

$p(E, c, F, G, F)$

(b) $p(X, Y, Y)$

$p(E, E, F)$

(c) $p(Y, a, b, Y)$

$p(c, F, G, F)$

(d) $ap(F0, c(b, c(B0, L0)), c(a, c(b, c(b, c(a, emp))))$

$ap(c(H1, T1), L1, c(H1, R1))$

12.11 列出下面知识库的所有基原子逻辑结论:

$q(Y) \leftarrow s(Y, Z) \wedge r(Z)$

$p(X) \leftarrow q(f(X))$

$s(f(a), b)$

$s(f(b), b)$

$s(c, b)$

$r(b)$

12.12 考虑下面的逻辑程序:

$f(empty, X, X)$

$f(cons(X, Y), W, Z) \leftarrow$

$f(Y, W, cons(X, Z))$

给出每一个自顶向下导出, 指明下面查询的替换(如例 12-31 所示):

$ask f(cons(a, cons(b, cons(c, empty))), L, empty)$

其全部的解答有哪些?

12.13 考虑下面的逻辑程序:

$rd(cons(H, cons(H, T)), T)$

$rd(cons(H, T), cons(H, R)) \leftarrow$

$rd(T, R)$

给出一个自顶向下导出, 指明下面查询的所有替换:

$ask rd(cons(a, cons(cons(a, X), cons(B, cons(c, Z)))), W)$

根据上面的导出, 该查询的解答是什么?

存在第二个解答吗? 如果存在, 给出导出过程; 如果不存在, 请解释原因。

12.14 考虑下面的逻辑程序:

$ap(emp, L, L)$

$ap(c(H, T), L, c(H, R)) \leftarrow$

$ap(T, L, R)$

$adj(A, B, L) \leftarrow$

$ap(F, c(A, c(B, E)), L)$

(a) 对于下面查询的一个解答, 给出自顶向下导出(包括所有的替换):

$ask adj(b, Y, c(a, c(b, c(b, c(a, emp)))))$

(b) 存在其他的解答吗? 如果存在, 请解释在上述解答的导出过程的哪一步可做出不同的选择, 并继续算法, 得到另一个解答。如果不存在另一个解答, 请解释为什么没有。

(在这个练习中, 你将作为一台计算机, 根本不用考虑符号的含义。如果你想为这个程序赋予一定的意义, 你可以将 ap 读作 $append$ 、 c 读作 $cons$ 、 emp 读作 $empty$ 、 adj 读作 $adjacent$ 。)

12.15 本题的目的是练习编写简单逻辑程序。

(a) 编写一个关系 $remove(E, L, R)$, 当 R 是从列表 L 中删除 E 的一个实例后剩余的列表时返回为真。如果 E 不是列表 L 的成员则该关系为假。

(b) 给出下面查询的所有解答:

$ask remove(a, [b, a, d, a], R)$

$ask remove(E, [b, a, d, a], R)$

$ask remove(E, L, [b, a, d])$

$ask remove(p(X), [a, p(a), p(p(a)), p(p(p(a))]), R)$

546

(c) 编写一个 *subsequence*(*L1*, *L2*) 关系, 如果列表 *L1* 中包含了 *L2* 中的元素的有序子集则为真。

(d) 对于下面的查询, 有多少不同的证明?

```
ask subsequence([a,d],[b,a,d,a])
```

```
ask subsequence([b,a],[b,a,d,a])
```

```
ask subsequence([X,Y],[b,a,d,a])
```

```
ask subsequence(S,[b,a,d,a])
```

请解释为什么有那么多?

12. 16 在本题, 你要为用户视频演示设计方案编写一个限定子句知识库。

假设视频使用下面的关系进行标注:

```
segment(SegId,Duration,Covers)
```

其中 *SegId* 是视频片段的标识符。(在一个实际的应用中, 可根据它来抽取视频片段)。 *Duration* 是视频片段的时长(秒)。 *Covers* 是该视频所涵盖的主题列表。一个视频标注的示例数据库如下:

```
segment(seg0,10,[welcome])
```

```
segment(seg1,30,[skiing,views])
```

```
segment(seg2,50,[welcome,computationalintelligence,robots])
```

```
segment(seg3,40,[graphics,dragons])
```

```
segment(seg4,50,[skiing,robots])
```

一个演示是片段的一个序列。你可将演示表示为片段标识符的列表。

(a) 如果 *Segments* 是一个演示, 其总时长小于或等于 *Maxtime* 秒且 *MustCover* 中列出的主题均被该演示中的一个片段所涵盖, 则公理化谓词

```
presentation(MustCover,Maxtime,Segments)
```

为真。这个谓词的目标是设计一个演示, 该演示涵盖了一定数量的主题且总时长限定在给定的范围内。

例如, 查询

```
ask presentation([welcome,skiing,robots],90,Segs)
```

应该至少返回下面两个解答(其视频片段顺序可能不同):

```
presentation([welcome,skiing,robots],90,[seg0,seg4])
```

```
presentation([welcome,skiing,robots],90,[seg2,seg1])
```

设为所有的符号给定了特定的解释且假设你已经在 *AILog* 或 *Prolog* 中测试了你的公理化(包括为你的查询找出所有的解答)。请解释为什么每一个答案是一个解答。

547

(b) 假设你有一个很好的用户接口, 有方法实际地查看该演示, 则请列出至少三个你想要但前面的程序没有提供的东西。(此问没有标准答案。你必须有创造力才能得满分。)

12. 17 构建一个知识库和字典(在图 12-11 的基础上)来回答诸如图 1-2 中的地理问题。对于每一个查询, 解释其是如何得到解答的, 或解释为什么难以使用本章介绍的工具得到解答。

548

本体和基于知识的系统

在发展一个能满足需要的计算理论的道路上，最重要的问题是发展与语义化相称的本体化。这并不是说语义问题消失了；语义仍像以往那样具有挑战性。语义和本体通过更高要求的本体问题联系起来，并继续像以往那样位于舞台的中央。

——Smith[1996, p. 14]

如何表达世界中现存的知识，使其容易被获取、调试、维护、传递、共享和推理？本章探讨如何明确说明智能 Agent 中符号的意义，如何在基于知识的调试和解释中使用这些意义，最后介绍 Agent 如何表示自己的推理以及如何利用这些表示来构建基于知识的系统。正如在前面引文中 Smith 所指出的，本体问题是构建智能计算 Agent 的中心问题。

13.1 知识共享

有一个恰当的表达只是建立一个基于知识的 Agent 的一部分工作。我们应该确保能够从多种不同来源获得知识，并且保证多种不同渠道来源的知识间可以互操作。此外还需要确保知识可以被用于有效的推理。

回想一下，**本体**是关于信息系统中符号的含义的规范说明。这里，信息系统是一个知识库或某种信息源，如温度计可被看做是一种信息系统。有些时候，符号的含义仅存在于知识库设计者的脑海中，或者存在于知识库的注释中。现在越来越常见的方式是，这些含义的规范说明以机器可解释的形式存在。这个形式的规范说明对于**语义互操作性**(semantic interoperability)——即不同知识库在一起工作的能力——是重要的。

【例 13-1】 采购 Agent 应该能够判断网站中所宣称的便宜的“chips”指代的是薯片、芯片、木条还是扑克筹码。本体可以为网站的术语提供明确意义。与仅使用符号“chip”不同，遵循了本体的网站可能使用符号“WoodChipMixed”——该符号由某一发布了本体的特定机构定义。通过使用这个符号并声明其来源于哪个本体，可以有效地避免使用“chips”时产生的歧义。网页的正式表示可能使用“WoodChipMixed”，而该符号可被翻译为简单的英语符号“chip”。如果另一个信息源使用的是符号“ChipOfWood”，则某一第三方可能会声明在该信息源中的术语“ChipOfWood”对应于我们的“WoodChipMixed”，从而使得两信息源可以相互结合。

在讨论如何指定本体之前，我们先讨论如何使用前一章的逻辑(可带有变量、项和关系)来建立灵活的表示。这些灵活的表示允许增加知识模块，包括在关系中添加变元。

通过为符号的含义设定一个规范说明，Agent 就可以在知识层面的知识获取、解释和调试中使用这个含义。

13.2 灵活的表示

本章的第一部分讲述如何使用逻辑工具建立灵活的表示方法。这些灵活的表示是现代本体论的基础。

13.2.1 选择个体和关系

给定一个逻辑表示语言(如前一章中开发的语言)及一个推理的世界,则知识库的设计者必须选择引用这个世界中的哪些内容。也就是说,他们必须选择要引用哪些个体和关系。乍看起来,他们可以引用在考察的世界中存在的那些个体和关系。然而,世界本身并没有提供关于个体的信息。为本世界建模的人创造了如何划分世界为个体。建模者将世界划分为小的部分,从而使得 Agent 能引用该世界中对于手中工作有意义的各部分。

【例 13-2】 在世界中,你可能认为“红色”是事物的合理属性。之所以这样,是因为你可能想告知传送机器人去拿红色的包裹。在世界中,物体的表面会吸收一些频率的光并反射其他频率的光。在一些应用中,一些用户可能称一些特定的反射特性为“红”。而领域的另外一些建模者可能使用另外一个频谱映射,并使用术语粉红、猩红、宝石红和深红,甚至另有一些建模者可能会对颜色频谱做不同的划分,以至于在任何语言中都无法找到对应的词语来描述,但是这种划分对于区分个体的不同类别非常有用。

正如建模者决定要表示哪些个体一样,他们还需要选择要使用的关系。下面通过一系列的例子来说明如何选择关系和个体的实用性指导原则。

【例 13-3】 假设你认为“red”是一个为个体分类的合适的属性,则可以把 *red* 作为一元关系,那么一个红色包裹 *a* 可以写做:

red(a)

如果用这种方式表示颜色信息,那么就可以容易地发出“什么是红色的”询问:

? *red(X)*

返回的 *X* 就是红色的个体。

在这种表示方法中,不能询问“包裹 *a* 是什么颜色?”。因为在限定子句的句法中,你不能提出下面的提问:

ask *X(a)*

因为在基于一阶逻辑的语言中,谓词的名字不能是变量。在二阶或高阶逻辑中,这可返回 *a* 的任何属性,而不仅仅是它的颜色。

有其他替换的方法允许你询问包裹 *a* 的颜色。在世界中没有任何东西会强制你设置一个谓词 *red*。你可简单地说颜色也是个体,且你可用常量 *red* 来表示红颜色。如果 *red* 是一个常量,你可以使用谓词 *color*, 其中 *color(Ind, Val)* 表示物理个体 *Ind* 拥有颜色 *Val*。“包裹 *a* 是红色的”现在可以写成:

color(a, red)

你所做的是重新审视这个世界:现在的世界由你可命名的个体组成,其中颜色也是个体。此时在物理个体和颜色之间有一种新的二元关系 *color*。根据这种新的表达关系你就可以使用下面的查询来询问“物体 *a* 是什么颜色的?”:

?*color(a, C)*

将一个抽象概念转化为一个对象被称为具体化(reify)。在前面的例子中,我们具体化了颜色 *red*。

【例 13-4】 前面例子的表示方式好像没有什么缺点。以前能够做的事情现在仍然能够做。与书写 *red(X)* 相比,书写 *color(X, red)* 并没增加多少困难,但是你现在可以询问事物的颜色了。因此,就会产生这样一个问题:你是否可以对每一个关系做类似的处理?要

做到何种程度?

正如对谓词 *red* 的分析一样, 你可以对例 13-3 中的谓词 *color* 做一个类似的分析。将 *color* 表示为一个谓词, 就不能提出“包裹 *a* 的哪个属性具有值 *red*?”这种问题, 此问题的合理解答是“*color*”。在例 13-3 中做一个类似的转换, 即你可以将如 *color* 的属性视作个体, 并创建一个关系 *prop*, 则“个体 *a* 的 *color* 属性的值为 *red*”可表示如下:

```
prop(a,color,red)
```

这种表示支持本例和上一个例子中所有的查询。你不必进一步做上述的分析了, 因为你能够使用 *prop* 关系来编写所有的关系。

个体-属性-值(individual-property-value)表示法是用单一的关系 *prop* 实现的, 其中:

```
prop(Ind,Prop,Val)
```

表示个体 *Ind* 的属性 *Prop* 的值是 *Val*。这被称为三元组表示(triple representation), 因为所有的关系都被表示为三元组(triple)。将三元组看做一个简单的由三个词组成的句子, 则其第一个元素称为主语(subject), 第二个元素是动词(verb), 第三个元素是宾语(object)。

三元组中的动词是一个属性(property)。属性 *p* 的域是个体的集合; 当 *p* 是一个动词时, 则这些个体能够作为三元组中的主语。属性 *p* 的范围是值的集合; 当 *p* 是动词时, 这些值可作为三元组中的宾语。

一个特性(attribute)是一个属性-值对。例如, 包裹的一个特性可能为其颜色是红色。如果两个包裹有相同的特性, 即它们的属性的值完全相同, 则此两个包裹相同。

对于三元组表示法, 有一些谓词可能看起来过于简单。

【例 13-5】 在将谓词 *parcel(a)*——其含义是 *a* 是一个包裹——转换为三元组的表示形式时, 看起来没有合适的属性和值。这里有两种将此谓词转换为三元组表示的方法。第一种是具体化包裹概念来说明 *a* 是一个包裹:

```
prop(a,type,parcel)
```

这里, *type* 是一个特殊的属性, 该属性建立个体与其类别间的联系。常量 *parcel* 表示类别, 该类别集合中的所有事物, 包括实在的和潜在的, 都是包裹。上述三元组表示个体 *a* 属于 *parcel* 类别。

第二种表示方法是将包裹作为一个属性, 此时“*a* 是一个包裹”可记为:

```
prop(a,parcel,true)
```

在这个表示中, *parcel* 是一个布尔属性, 当某事物为包裹时, 该事物的 *parcel* 属性值为真。

布尔属性(Boolean property)是一种属性, 其取值范围是{*true*, *false*}, 其中 *true* 和 *false* 是语言中的常量符号。

对于三元组表示法来说, 有些谓词可能看起来过于复杂。

【例 13-6】 假设你希望表示如下的关系:

```
scheduled(C,S,T,R)
```

其含义是课程 *C* 的 *S* 节计划于时刻 *T* 在房间 *R* 开始。例如, “课程 *cs422* 的第 2 节将于 10:30 在房间 *cc208* 开始”可记为:

```
scheduled(cs422,2,1030,cc208)
```

为用三元组表示该关系, 你可以创建一个新的个体 *booking*(预订)。因此, *scheduled* 关系就可具体化为一个预订个体。

一个预订有多个属性，即一个课程、课程的一节、一个开始时间和一个房间。为表示“课程 cs422 的第 2 节计划于 10:30 在房间 cc208 开始”，你可命名该预订，如一个常量 b123，并可记为：

```
prop(b123,course,cs422)
prop(b123,section,2)
prop(b123,start_time,1030)
prop(b123,room,cc208)
```

这种新的表示法具有许多优点。最重要的优点是其模块化结构；容易看出哪个值从属于哪个属性。也容易添加新属性，如授课的教师或者授课的时间长度。使用这个新的表示法，可容易地添加如“Fran 讲授课程 cs422 的第 2 节，计划于 10:30 在房间 cc208 开始”或课程时间是 50 分钟：

```
prop(b123,instructor,fran)
prop(b123,duration,50)
```

如果使用 *scheduled* 谓词，则很难添加授课教师或者课程持续时间，因为这需要为每一个谓词实例添加一个额外的论元。

553

13.2.2 图形化表示

你可以使用图来解释 *prop* 关系，其中关系

```
prop(Ind,Prop,Val)
```

使用节点 *Ind* 和 *Val* 及该两节点间带有标签 *Prop* 的边来描述，称这种图为语义网络 (semantic network)。给定这样一个图形化表示，则有一个使用 *prop* 关系的直接到知识库的映射。

【例 13-7】 图 13-1 表示了传送机器人的一个语义网络，该网络表示了机器人可能有的适用于特定计算机的一类知识。该网络中表示的部分知识如下：

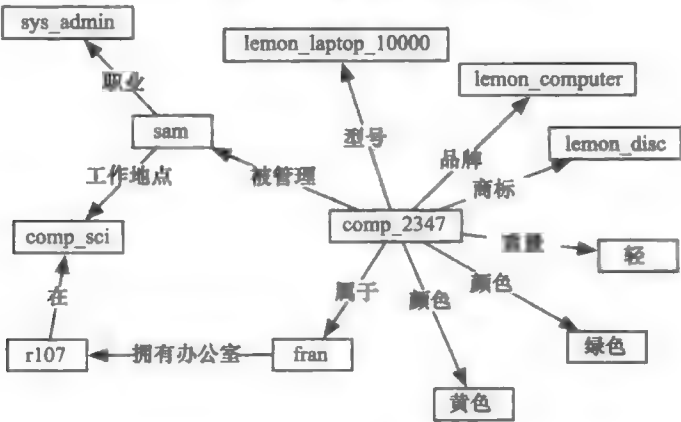


图 13-1 语义网

554

```
prop(comp_2347,owned_by,fran)
prop(comp_2347,managed_by,sam)
prop(comp_2347,model,lemon_laptop_10000)
prop(comp_2347,brand,lemon_computer)
prop(comp_2347,has_logo,lemon_disc)
```

```
prop(comp_2347,color,green)
prop(comp_2347,color,yellow)
prop(comp_2347,weight,light)
prop(fran,has_office,r107)
prop(r107,in_building,comp_sci)
```

554

该网络也展示了知识是如何组织的。例如，可很容易地看出编号为 2347 的计算机被某人(Fran)拥有，此人的办公室(r107)在建筑物 *comp_sci* 中。图中明显的直接标定可被人类和机器使用。

这种图形符号具有多个优点：

- 不需要学习特定逻辑的句法就可以容易地看出相应的关系。图形符号可帮助知识库的建造者组织他们的知识。
- 你可以忽略节点的标签，这些标签只是些无意义的名字。例如，例 13-6 中的名字 *b123*，或图 13-1 中的 *comp_2347*。你可以把这些节点视为空白；如果你必须将它们映射到逻辑形式，则可以为它们任意指定一个名字。

三元组的简明语言

Turtle 是表示三元组的简单语言。它是为语义网络而创造的语言之一。它也用于资源描述框架(Resource Description Framework, RDF)的句法分析——其中 RDF 是从一个相似的语言 Notation 3 或 N3 扩展而来。

在 Turtle 和 RDF 中，任何事物——包括个体、类别和属性——都是资源。统一资源标识符(Uniform Resource Identifier, URI)可以用来唯一地标识任何物体。URI 标记写在尖括号内，它往往有一个 URL 的形式，因为每个 URL 都是唯一的。例如，`<http://aispace.org>` 可以是一个 URI。在 URI 中，“#”表示指向网页中的个体。例如，`<http://cs.ubc.ca/~poole/foaf.rdf#david>` 表示网页 `http://cs.ubc.ca/~poole/foaf.rdf` 中的个体 *david*。URI`<>` 表示当前文档，因此，URI`<#comp_2347>` 表示一个在当前文档中定义的个体。

三元组可以简单地记为：

Subject Verb Object

其中，*Subject* 和 *Verb* 都是 URI，而 *Object* 可以是一个 URI 或一个文字(字符串或数字)。 *Verb* 指示属性。 *Object* 是 *Subject* 的属性 *Verb* 的值。

【例 13-8】 例 13-7 中的三元组可用 Turtle 写为如下形式：

```
<#comp_2347><#owned_by><#fran>.
<#comp_2347><#managed_by><#sam>.
<#comp_2347><#model><#lemon_laptop_10000>.
<#comp_2347><#brand><#lemon_computer>.
<#comp_2347><#has_logo><#lemon_disc>.
<#comp_2347><#color><#green>.
<#comp_2347><#color><#yellow>.
<#fran><#has_office><#r107>.
<#r107><#serves_building><#comp_sci>.
```

555

其中的标识符“*fran*”不代表个体的名称。如果希望表示某人的名字叫 Fran，则可记为：

```
<#fran><#name>"Fran".
```

在 Turtle 语言中有一些常用的缩略语。逗号分组具有相同的主语和谓词的对象，即 $S \ V \ O_1, O_2$ 。

是如下表示的简写：

$S \ V \ O_1$ 。

$S \ V \ O_2$ 。

分号分组具有相同主语的动词-宾语对，即

$S \ V_1 \ O_1; V_2 \ O_2$ 。

是下面的简写：

$S \ V_1 \ O_1$ 。

$S \ V_2 \ O_2$ 。

方括号用于定义一个没有标识符的个体。该无名称的资源可用于某三元组的宾语，但不能被引用。逗号和分号均可用于表示这类资源的属性。因此，

$[V_1 \ O_1; V_2 \ O_2]$

表示一个体，其属性 V_1 的值为 O_1 ，属性 V_2 的值为 O_2 。这种无名称个体的描述有时被称为框架(frame)。动词有时被称为槽(slot)，而宾语被称为填充物(filler)。

【例 13-9】 下面的 Turtle 语句

```
<comp_3645><#owned_by><#fran>;
    <#color><#green>,<#yellow>;
    <#managed_by>[<#occupation><#sys_admin>;
        <#serves_building><#comp_sci>].
```

表明<#fran>拥有<comp_3645>，其颜色是绿色和黄色，且其被一个资源所管理，而该资源的工作是系统管理、工作地点是 *comp_sci*。

556

其是如下三元组的简写：

```
<comp_3645><#owned_by><#fran>.
<comp_3645><#color><#green>.
<comp_3645><#color><#yellow>.
<comp_3645><#managed_by><i2134>.
<i2134><#occupation><#sys_admin>.
<i2134><#serves_building><#comp_sci>.
```

不过，其中的 URI<i2134>不能在任何地方被引用。

对于读者来说，其很难知道设计者设计一个特定的 URI(如<#name>)时的意图；同样也很难知道这个术语的使用是如何与其他人对同一术语的使用发生关联的。然而，人们已经对一些特定术语的含义达成了共识。例如，属性<http://xmlns.com/foaf/0.1/#name>是对一个对象的名称的标准定义。因此，如果我们编写如下语句：

```
<#fran><http://xmlns.com/foaf/0.1/#name>"Fran".
```

则我们的意思是特定的 *name* 属性就遵循上述定义。

URL <http://xmlns.com/foaf/0.1/>中的内容无关紧要，只要使用 URI<http://xmlns.com/foaf/0.1/#name>的人的意图均是要表达同一个属性就可以了。一个 URL，在编写它时，仅是重定向到一个网页。然而，“friend of a friend”项目(这正是“foaf”的含义)使用该命名空间来表达某些事情。这样做可使事情变得简单些，因为人们都在这样使用。

在 Turtle 中, 通过使用 "@prefix" 声明, 可以使用简化的 "name:" 来替换一个 URL 和尖括号。例如, 作如下声明:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/#>
```

则 "foaf: name" 就表示了 <http://xmlns.com/foaf/0.1/# name> 的简写形式。类似的, 有如下声明

```
@prefix: <#>
```

我们就可用: color 表示 <# color>。

在 Turtle 中也允许为没有具体化的函数的参数加括号。它还允许使用缩写 "a" 表示 "rdf: type", 但我们在这里不遵循这一惯例。

13.2.3 原始关系与导出关系

通常, 你对域的了解要比对事实库的了解更多; 你知道一些一般规则, 根据这些规则你可以导出其他的事实。哪些事实需要明确给出, 哪些事实可以使用规则导出, 这是设计和建造知识库时需要做出的选择。

557

原始知识 (primitive knowledge) 是使用事实明确定义的知识。**导出知识** (derived knowledge) 是可从其他知识中推断出来的知识。导出知识通常使用规则来明确规定。

规则的使用可以使知识的表示更加简练。导出关系允许从对域的观察中得出结论。这是很重要的, 因为你不会直接观察到域中的所有事情。对域的了解大多来自以观察和一般知识为基础的推理。

知识库与面向对象程序设计中的类

基于知识的系统中的“个体”与“类别”的使用与**面向对象程序设计** (object-oriented programming, OOP) 语言 (如 Smalltalk 和 Java) 中的“对象”和“类”的使用非常相似。对此你不必过于惊讶, 因为它们在历史上就相互联系。它们之间有重要的不同, 如果直接比拟它们, 那么这些不同往往会造成更多的困扰而不是提供更多的帮助:

- OOP 中的对象是可以计算的对象; 它们是数据结构和相联系的程序。Java 中的对象 "person" 不是一个真实的人。然而, 知识库 (KB) 中的个体是现实世界中的 (有代表性的) 物体。KB 中的个体 "person" 可以是一个真实的人。个体 "chair" 可以是一个真实的椅子, 你可以坐在上面; 如果你撞到它, 这个 "chair" 可以伤到你。你可以给 Java 中的 "chair" 对象发送消息, 并从它获得答案, 然而现实世界中的椅子并不理会你对它说的话。KB 通常不是用来与椅子进行交互, 而是用来推理关于椅子的信息的。真实的椅子只会停留在那里, 直到它被一个物理的 Agent 移动了才会离开原处。
- 在 KB 中, 一个对象的表示仅仅是在一个 (或几个) 抽象水平上的近似。真实的对象往往要比表示出的复杂得多。我们通常不去表示一个椅子中的纤维个体。在 OOP 系统中, 则仅有表示出的对象的属性。系统能够知道关于一个 Java 对象的所有情况, 而不知道真实的个体。
- Java 中的类结构用来表示设计的对象。系统分析员或程序员来创建一个设计。例如, 在 Java 中, 一个对象仅是最低级别的一个类成员。在 Java 中没有多重继承的概念。真实对象并没有很好地得到表达。一个人可以是一个足球教练、一个数学家或是一位母亲。

- 计算机程序使用的数据结构不能有不确定性；它必须选择使用特定的数据结构。然而，现实世界中的事物的类型对于我们来说具有不确定性。
- 实际中，KB 中的模型表示不能执行任何操作。在 OOP 系统中，对象完成计算工作。在 KB 中，仅仅是一个表示，即其仅是引用了世界中的一个对象。
- 尽管面向对象建模语言，如 UML，可用于表示 KB，但这并不一定是最佳选择。一个好的 OO 建模工具提供了一些功能以帮助构建好的设计。然而，我们要建模的世界可能根本没有一个好的设计。试图为一个凌乱的世界构建一个好的设计模式可能没有什么益处。

一个使用导出知识的标准方法是将个体划分为类。我们将通用属性赋予类，从而使得个体可以继承类的属性。我们将个体划分为类的原因是类的成员要么具有共同的特性，要么具有共同且有意义的属性(参考后文的“亚里士多德定义”)。

类(class)是那些实际的或潜在的属于该类的个体的集合。逻辑上，这是一个**内涵(intensional)**集合，由一个**特征函数(characteristic function)**定义，如果一个个体属于该类则其特征函数为真，否则为假。另一个可选方法是**外延(extensional)**集合，其由属于该类的成员列表定义。

例如，“椅子”类是所有可称为椅子的事物的集合。我们不希望将椅子的事物作为该集合的定义，因为尚未做好的椅子也属于椅子类。我们不希望两个类仅因为有相同的成员而等价。例如，绿色的独角兽类和高度恰好为 124 米的椅子类是不同的类，即使它们含有相同的元素；这两个集合均为空。

依据类的定义，任何可以被描述的集合均可成为一个类。例如，包含数字 17 的集合、伦敦塔和凯撒大帝的左脚都可以是一个类，但这些类不是很有用。一个**自然类(natural kind)**是使用该类比不使用该类可更简洁地描述事物的那些类。例如，“哺乳动物”是一个自然类，对哺乳动物共有特性的描述形成一个知识库，使用这个知识库要比不使用知识库、重复描述哺乳动物的共同特性更加简洁。

我们使用 *type* 属性来表示“是一个类的成员”。从而，在限定子句语言中有：

prop(X, type, C)

表示个体 *X* 是类 *C* 的一个成员。

创建 RDF 和 RDF 模式的人使用的属性正是我们在这里为表达类的成员关系所需要的属性。在 Turtle 语言中，我们可以定义如下缩写：

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

给定上述声明，*rdf:type* 表示属性 *type*，该属性将个体与其所属的类关联起来。通过引用 URI 中的 *type* 的定义，这就变成了一种标准定义，其能够被他人所用，且能区分于“*type*”的其他含义。

类间的 *rdfs:subClassOf* 属性指明了一个类是另一个类的子集。在 Turtle 中

S rdfs:subClassOf C.

表示类 *S* 是类 *C* 的子类。从集合的角度来说，这就意味着 *S* 是 *C* 的一个子集。也就是说，类 *S* 中的每一个体都是类 *C* 中的个体。

【例 13-10】 在例 13-7 中明确规定了计算机 *comp_2347* 的商标是一个柠檬盘(lemon disc)。你可能知道所有的 Lemon 牌计算机都使用这个商标。因此，另一种表示方法是将该商标与 *lomon_computer* 联系起来，然后再导出 *comp_2347* 的商标。这个表示法的优点

在于当你发现另一台 Lemon 牌的计算机时，你可以推导出它的商标。

在 Turtle 中，语句

```
:lemon_computer rdfs:subClassOf:computer.
:lemon_laptop_10000 rdfs:subClassOf:lemon_computer.
:comp_2347 rdf:type:lemon_laptop_10000.
```

说明一个 lemon 牌计算机是一种计算机，一个 lemon laptop 10000 是一个 lemon 牌计算机，及 comp_2347 是一个 lemon laptop 10000 型的计算机。一个扩展的例子如图 13-2 所示，其中带阴影的矩形表示类，从类发出的箭头不是类的属性而是类成员的属性。

560

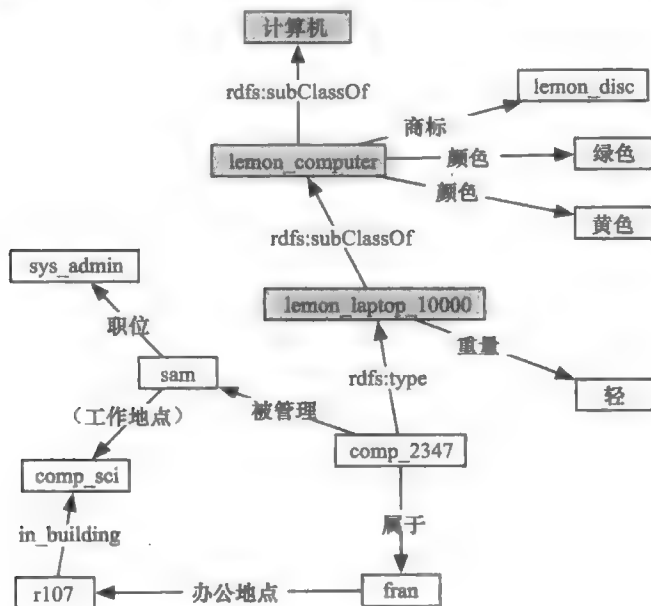


图 13-2 允许继承的语义网络

类型和子类之间的关系可以写成一个限定子句的形式：

```
prop(X,type,C)←
    prop(X,type,S) ∧
    prop(S,subClassOf,C)
```

你可以把 *type* 和 *subClassOf* 看做是一种允许属性继承(property inheritance)的特殊属性。属性继承是指在类的层次上确定了一个属性的值时，类的成员能够继承它。如果类 *c* 的所有成员的属性 *p* 的值为 *v*，则在 Datalog 中可写为：

```
prop(Ind,p,v)←
    prop(Ind,type,c).
```

该规则与前面提到的将类型与子类型联系起来的规则一起可用于属性的继承。

【例 13-11】 所有 lemon 牌计算机都有一个柠檬盘作为商标，其颜色为黄色和绿色(参见图 13-2 中的 *logo* 和 *color* 箭头)。这可使用下面的 Datalog 程序来表示：

```
prop(X,has_logo,lemon_disc)←
    prop(X,type,lemon_computer).
prop(X,color,green)←
    prop(X,type,lemon_computer).
prop(X,color,yellow)←
```

`prop(X,type,lemon_computer).`

可从上述子句中导出的 `prop` 关系本质上与可从图 13-1 中的无层次的语义网络中导出的结果一致。通过该结构化的表示法，在增加一个新的 Lemon Laptop 10000 型计算机时，你只需声明其为一个 Lemon Laptop 10000 型计算机，而其颜色和商标属性均可通过继承关系导出。

RDF 和 Turtle 中没有限定子句。在这些语言中，类的成员关系不表示为谓词，而是表示为集合。为表示集合 S 的所有元素对于谓词 p 都取值 v ，我们可以说 S 是对于谓词 p 来说其值为 v 的所有事物的一个子集。

【例 13-12】 为说明所有的 lemon 牌计算机都有一个柠檬盘商标，我们说 lemon 牌计算机是属性 `has_logo` 的取值为 `lemon_disc` 的所有事物集合的一个子集。

这个内容的一个表示如图 13-3 所示。其中，`:computer` 和 `:logo` 都是类。`:lemon_disc` 是类 `:logo` 的成员。`:has_logo` 是一个属性，其域为 `:computer`，其值域是 `:logo`。`:lemon_computer` 是 `:computer` 的一个子类。它也是 `:has_logo` 属性为 `:lemon_disc` 的所有个体的集合的一个子类。

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <#>.

:computer    rdf:type    rdfs:Class.
:logo        rdf:type    rdfs:Class.
:lemon_disc  rdf:type    :logo.
:has_logo    rdf:type    rdf:Property ;
              rdfs:domain :computer ;
              rdfs:range  :logo.
:lemon_computer
  rdf:type    rdfs:Class ;
  rdfs:subClassOf :computer ;
  rdfs:subClassOf
    owl:ObjectHasValue(:has_logo :lemon_disc).
```

图 13-3 例 13-12 的 Turtle 表示

`owl:ObjectHasValue` 是 OWL(参见下面的解释)类的构造函数，从而 `owl:ObjectHasValue(:has_logo:lemon_disc)` 是属性 `:has_logo` 的值为 `:lemon_disc` 的所有个体的类。

对于决定哪些应是原始的，哪些应是导出的，有如下一些有用的一般原则：

- 当将一个特性关联到一个体时，选择最一般的类 C ，使得该个体属于类 C 且类 C 的所有成员均有该特性，同时该特性与类 C 相关联。可以使用继承来为该个体和类 C 的所有成员导出特性。这种表示方式往往能使知识库更加简洁，并且意味着可以容易地添加新个体，原因是这些新个体如果属于类 C 则它们就自动地继承了该类的特性。
- 不要将类的或有特性关联到类上。**或有特性**(contingent attribute)是其值随环境的变化而变化的一类特性。例如，在当前的情况下，有可能购买的所有的计算机都装在棕色盒子中。然而，将此作为一个特性关联到 `computer` 类却不是一个好主意，因为对于购买的其他的计算机这一点不一定成立。

- 公理化的因果方向。如果要在原始原因和原始结果间做出选择，则选择原始原因。这样做，在时域发生变化时，信息会更可能保持稳定。

13.3 本体与知识共享

构建大型的基于知识的系统是很复杂的，原因如下：

1) 知识往往来自多种渠道，需要集成。此外，这些知识来源可能对世界做了不同的划分。通常情况是，知识来自不同的领域，而这些领域各自有独特的术语且根据自己的需要对世界做了划分。

2) 系统随着时间的推移而演化，很难预测出在将来应作出的明显差别。

3) 参与设计知识库的人必须选择要表示出哪些个体和关系。世界并不是划分为个体；对世界进行划分由智能 Agent 完成，这也是其理解世界的方式。设计知识库的人应该在对世界的划分上达成一致。

4) 通常，记住自己定义的符号的含义就很难了，更不用说去弄清楚他人定义的符号的含义了。这一点有两方面的含义：

- 给出一个在计算机中使用的符号，确定其含义；
- 给出某人头脑中的一个概念，确定应该使用哪个符号来表示它；也就是说，确定该概念是否已经使用，如果已经使用，则弄清楚其当前的符号表示。

为了共享和传递知识，制定一个共同的词汇表并统一词汇的含义是十分重要的。

概念化(conceptualization)是在计算机使用的符号(即词汇)与世界中的个体和关系之间的映射。它为世界提供了一个特定的抽象，并为该抽象提供了符号表示。对于小型知识库，这种概念化可以存在于设计者的头脑中，或者可以使用自然语言在文档中给予确定。这种非形式化的概念化表示不适用于大型系统——在大型系统中必须共享这种概念化。

在哲学范畴中，**本体论**(ontology)研究的是存在什么。在人工智能范畴中，**本体**是一个信息系统中符号含义的规范。也就是说，本体是一个概念化的规范。它是假设存在哪些个体和关系以及与之相关的术语的规范。通常，它指定个体可被归于哪些类型，将使用哪些属性，并给出一些限制词汇使用的公理。

【例 13-13】 关于个体的能够出现在地图上的本体可以明确符号“ApartmentBuilding”表示公寓楼。本体并不定义一座公寓楼，但它会对公寓楼有足够的描述从而使得其他人能够理解这个定义。我们希望那些倾向于使用不同符号的人能够使用该本体来找到合适使用的符号(参见图 13-4)。多个人能够一致地使用这些符号。本体还应能让人们检查符号的含义。也就是说，对于一个概念，他们希望能够找到对应的符号并使用那个符号，且他们希望能够确定其含义。

本体中可能给出了一些公理来限制某些符号的使用。例如，本体可以指定公寓楼是大楼，而大楼是人工建造的物体。它可能给出了大楼大小的限制，从而使得鞋盒和城市都不属于大楼。本体也可能说明一个大楼不能同时出现在两个地理上分散的地点(所以如果你把大楼的一部分移动到不同的位置，则该大楼已经不再是一个单一的大楼)。因为公寓楼是大楼，所以上述对大楼的限制也适用于公寓楼。

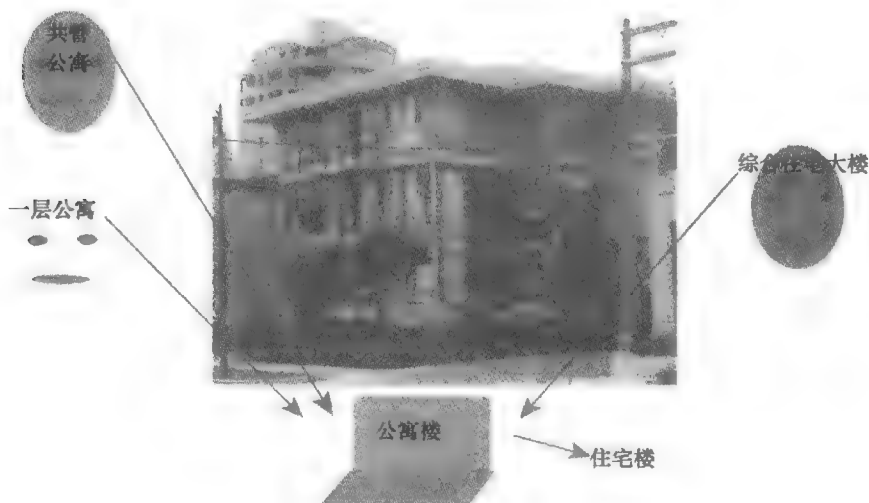


图 13-4 从概念化映射到符号

语义网

语义网(semantic web)是允许将机器可解释的知识分布到万维网上的一种方法。网站可提供被计算机使用的信息,而不是仅能提供让人阅读的 HTML 页面。

在最基本的层次上,XML(the Extensible Markup Language)提供了一个为机器的阅读而设计的句法,且该句法也可供人阅读。它是一个基于文本的语言,其中的项使用分层的方式来标注。XML 的句法可以很复杂,但在最简单的层面上,标注或者以<tag.../>的形式存在,或者以<tag...>...</tag>的形式存在。

URI(Uniform Resource Identifier)用来唯一标识一个资源。**资源**是可以被唯一标识的任何事物。一个 URI 是指示了一个资源的字符串,如一个网页、一个人或一个公司。URI 通常使用网络地址的句法。

RDF(Resource Description Framework)是一个建立在 XML 上的语言,提供了个体-属性-值的三元组。

RDF-S(RDF Schema)允许你使用其他资源(如使用 *subClassOf*)来定义资源(也包括属性)。RDF-S 也可以限制域和属性的范围,并提供了容器(集合、序列及可选项——其中必有一个为真)。

RDF 允许句子使用其自己的语言具体化。这意味着它可以表示任意的逻辑公式,因此通常不可判定。不可判定未必是一件坏事;它只是意味着你不能确定计算可能需要的时间。带有函数符号的简单逻辑程序及几乎所有的编程语言都是不可判定的。

OWL(Web Ontology Language)是万维网的本体语言。它定义了具有固定解释的一些类和属性,这些类和属性可用于描述类、属性和个体。它有内置的判定个体、类和属性等价的机制,另外还有对域、属性范围和其他的属性约束的限制(如传递性和基数性)。

人们已对构建大型的通用本体做出了一些努力,如 cyc(www.cyc.com),不过,建立语义网的目标是将各个团体聚拢到本体上来。任何人都可以建立一个本体。想要建立一个知识库的人可以使用现有的本体,也可开发他们自己的本体——通常都是建立在现存本体的基础上。因为获得语义互操作性是知识库建立者的意愿,所以公司和个体应该尽量为他们的领域采用标准的本体,或者开发一个从他们的本体到其他本体之间的映射。

本体通常独立于特定的应用，并且本体的符号由多个开发社团共同确定。一个本体由下面几部分组成：

- 知识库中要描述事物类别的词汇(包括类和属性)；
- 类间的组织，如使用 `subClassOf` 或 `subPropertyOf`，或使用亚里士多德定义建立类间的层次结构；
- 限制一些符号的含义以更好地反映它们的含义的公理，如一些属性是传递的、域和范围是受限制的，或对个体的一个属性的取值个数的限制。有些时候，一些关系由更基本的关系来定义，但最终是建立在原始关系上。

本体并不描述在设计时尚不知道的个体。例如，描述建筑物的本体通常不包括实际的建筑物。本体可以描述固定的和应该共享的个体，如一周中的几天，或一些颜色。

【例 13-14】 考虑一个设计目的是寻找住宿的交易 Agent。用户可以使用这样的一个人 Agent 来描述他们期望的住宿条件。交易 Agent 可以搜索多个知识库以找到合适的住所，或在有合适的住所时通知用户。本体需要为用户明确符号的含义并允许知识库间的互操作。本体提供了一种媒介将用户需求和知识库连接在一起。

在这样的领域中，房屋和公寓楼可能都是居民楼。尽管推荐租一套房子还是推荐租一个公寓楼中的一套公寓应该很慎重，但为没有明确说明想租整个公寓楼的人，推荐租整个公寓楼却可以不那么慎重。一个“居住单元”可定义为一些共同生活的人居住的房间集合，也可能是房屋中介提供的出租单元。在某些时候，设计者必须决定一个房屋中的一个房间是一个居住单元，还是一个用于出租的共享房间的一部分是一个居住单元。通常，边界问题——那些最初没有预料到的问题——并没有明确刻画，但随着本体的演化而逐步有了好的定义。

本体将不包含对实际的房屋或公寓的描述，因为这些可住宿的房屋会随时间而改变，但是本体词汇的含义不会改变。

本体的主要目的是文档化符号的含义，即符号(计算机中的)与概念(在人类头脑中的)之间的映射。给定一个符号，人们能够使用本体来确定该符号的含义。当需要表示一个概念时，人们使用本体来查找合适的符号或者确定该概念在本体中尚不存在。第二个目的——该目的通过使用公理来实现——是允许推理或发现一些值的组合是不一致的。建立本体的主要挑战是组织要表达的概念以允许人们将这些概念映射到计算机中的符号上，且使得计算机能从表述的事实出发推理出有用的新知识。

亚里士多德定义

为对象分类——现代本体论的基础——有一个很长的历史。亚里士多德(公元前 350 年)提出使用下面的方式来给出类 C 的定义：

- 种类： C 的超类。
- 区别特征：区分类 C 的成员与超类 C 的其他成员的属性。

亚里士多德预见到了在定义中会出现的许多问题：

如果种类是不同的且相互并列，则它们的区别特征就是它们在类别上的不同。以“动物”种类和“知识”种类为例。“有脚”、“两只脚”、“有翅膀”、“水生”是“动物”种类的区别特征；知识类别却不能依靠上述区别特征来区分。知识的不同类别不能通过“两只脚”来区分。(亚里士多德，公元前 350 年)

注意，这里的“并列”指的是两方没有从属关系。

在现代的本体中，我们会说“动物”是一个类而“知识”是另一个类。属性“两只脚”的域

为“动物”。如果某事物是知识的一个实例，则它对于属性“两只脚”没有取值。

为根据**亚里士多德定义**(Aristotelian definition)建立一个本体，需要做下面的工作：

- 对每一个你想要定义的类，确定一个相关的超类，然后选出那些将该类区分于其他子类的特性。每一个特性提供了一个属性及其取值。
- 对于每个属性，定义一个最一般的且该属性对其来说有意义的类，然后定义该属性的域为此类。将该属性有意义的范围定义为一个类(可以使用枚举其值或使用亚里士多德定义的方法来定义该范围类)。

这个过程可以很复杂。例如，定义“奢华家具”时，你可能认为其超类是“家具”，其显著特点是成本高且柔软。家具的柔软和岩石的柔软是不同的。你还可能希望区分其与肌理的差别(它们可能都被认为是软的)。

这种方法通常不会给出一个层次结构。一个对象可以属于许多类。每一个类不是仅有一个最具体的超类。然而，这种结构仍然可以直接检查一个类是否是另一个类的子类、检查一个类的含义以及确定在你的头脑中与该类对应的概念。

在极少数情况下，按这种方法会构成一个树形结构，其中最著名的是针对生物界的**林奈分类**(Linnaean taxonomy)。这个分类是树形结构源于类似生物的进化过程。在其他领域强制使用树形结构一直不太成功。

567

13.3.1 描述逻辑

统一资源标识符具有一定的意义，因为有人发布它说它有意义，又因为人们根据这个含义来使用它。这是现在的方式，但我们想获得更多。我们希望获得一些含义从而允许计算机能在其上做一些推理。

现代本体语言，如 OWL，建立在**描述逻辑**(description logic)的基础上。描述逻辑用来描述类、属性和个体。描述逻辑的主要思想是分离如下内容：

- **描述术语的术语知识库**(terminological knowledge base)，其中术语应保持不变，尽管其被建模的域是可变的。
- **描述在一些域中、某些事件点时，哪些事物为真的断言知识库**(assertional knowledge base)。

通常，术语知识库在设计系统时定义，它又定义了本体，且它只在词汇含义变化时才会变化，而这一情况很少见。断言知识库通常包含了知识，其中的知识是与特定的情况紧密相关的，只在运行时才会知道这些知识。

通常使用三元组来定义断言知识库，而使用一种如 OWL 的语言来定义术语知识库。

OWL 通过下面几个方面来描述域：

- **个体**(individual)：在世界中被描述的事物(一个特定的房子或一个特定的预订都可以是个体)。
- **类**(class)：是个体的集合。类是所有实在的和潜在的属于该类的事物的集合。例如，“房子”类是所有可以归为“房子”的事物的集合，而不仅是我們感兴趣的域中存在的房子。
- **属性**(property)：用来描述个体。**数据类型属性**(datatype property)的值是基本数据类型，如整型或字符串。例如，“streetName”可能是街道和字符串之间的数据类型属性。**对象属性**(object property)的值是其他个体。例如，“nextTo”可能是两

个房子之间的属性，“onStreet”可能是房子和街道之间的属性。

OWL 有三个变种，其区别在于对类和属性施加的限制不同。在 OWL-DL 和 OWL-Lite 中，类不能是个体和属性，且属性也不能是个体。在 OWL-Full 中，个体、属性和类的类别不必是不相交的。OWL-Lite 中有一些句法限制，这种限制不影响含义，但可使推理更简单。

OWL 中没有唯一名字假设；两个名字不意味着它们指代了不同的个体或类。它也没有知识完备性假设；它不假设所有相关的事实均已被描述。

图 13-5 给出了一些基本的类和一些类的构造函数。该图使用集合符号来定义一个类中的个体集合。图 13-6 给出了 OWL 的基本谓词。前缀 owl: 表示来自 OWL。为在本体中使用这些属性和类，你可引入合适的 URI 的缩写形式：

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
```

下表是 OWL 中部分内置的类与类的构造函数，其中 C_i 表示类， P_i 表示属性， I_i 表示个体， n 是一个整数。
S 是集合 S 中元素的个数：

类	包含的内容
owl: Thing	所有个体
owl: Nothing	无个体(空集)
owl: ObjectIntersectionOf(C_1, \dots, C_k)	属于 $C_1 \cap \dots \cap C_k$ 的个体
owl: ObjectUnionOf(C_1, \dots, C_k)	属于 $C_1 \cup \dots \cup C_k$ 的个体
owl: ObjectComplementOf(C)	不属于 C 的个体
owl: ObjectOneOf(I_1, \dots, I_k)	I_1, \dots, I_k
owl: ObjectHasValue(P, I)	属性 P 的值为 I 的个体，即 $\{x: x P I\}$
owl: ObjectAllValuesFrom(P, C)	所有属性 P 的值属于 C 的个体，即 $\{x: x P y \rightarrow y \in C\}$
owl: ObjectSomeValuesFrom(P, C)	属性 P 的值部分属于 C 的个体，即 $\{x: \exists y \in C \text{ 满足 } x P y\}$
owl: ObjectMinCardinality(n, P, C)	通过属性 P 至少与 n 个 C 类的个体关联的个体 x ，即 $\{x: \# \{y \mid x P y \text{ 且 } y \in C\} \geq n\}$
owl: ObjectMaxCardinality(n, P, C)	通过属性 P 至多与 n 个 C 类的个体关联的个体 x ，即 $\{x: \# \{y \mid x P y \text{ 且 } y \in C\} \leq n\}$

图 13-5 OWL 的部分内置类和类的构造函数

在上述两个图中， xPy 是一个三元组。请注意，这意味着定义谓词的含义，而不是定义句法。谓词可以使用不同的句法，如 XML、Turtle 或传统的关系符号。

有一个属性构造函数：owl: ObjectInverseOf(P)，其是 P 的逆属性，即其为属性 P^{-1} ，满足 $yP^{-1}x$ 当且仅当 xPy 。请注意，这仅适用于对象属性；数据类型属性没有逆，因为数据类型不能作为三元组的主语。

上面两个图中列出的类和陈述并不完整。对于数据类型属性，具有对应的合适的数据类型类。例如，owl: DataSomeValuesFrom 和 owl: EquivalentDataProperties 对于数据类型属性，其相应的对象符号具有相同的定义。在 OWL 中，还有其他的构造方法来定义属性、评论、注释、版本控制，及导入其他本体。

OWL 有下面的具有固定解释的谓词，其中 C_i 是类， P_i 是属性， I_i 是个体； x 和 y 是全称量化变量。

描述	含义
<code>rdf:type(I, C)</code>	$I \in C$
<code>rdfs:subClassOf(C_1, C_2)</code>	$C_1 \subseteq C_2$
<code>owl:EquivalentClasses(C_1, C_2)</code>	$C_1 \equiv C_2$
<code>owl:DisjointClasses(C_1, C_2)</code>	$C_1 \cap C_2 = \{\}$
<code>rdfs:domain(P, C)</code>	如果存在 xPy ，则 $x \in C$
<code>rdfs:range(P, C)</code>	如果存在 xPy ，则 $y \in C$
<code>rdfs:subPropertyOf(P_1, P_2)</code>	xP_1y 蕴含 xP_2y
<code>owl:EquivalentObjectProperties(P_1, P_2)</code>	xP_1y ，当且仅当 xP_2y
<code>owl:DisjointObjectProperties(P_1, P_2)</code>	xP_1y 蕴含非 xP_2y
<code>owl:InverseObjectProperties(P_1, P_2)</code>	xP_1y ，当且仅当 yP_2x
<code>owl:SameIndividual(I_1, \dots, I_n)</code>	$\forall j \forall k I_j = I_k$
<code>owl:DifferentIndividuals(I_1, \dots, I_n)</code>	$\forall j \forall k j \neq k$ 蕴含 $I_j \neq I_k$
<code>owl:FunctionalObjectProperty(P)</code>	如果 xPy_1 且 xPy_2 ，则 $y_1 = y_2$
<code>owl:InverseFunctionalObjectProperty(P)</code>	如果 x_1Py 且 x_2Py ，则 $x_1 = x_2$
<code>owl:TransitiveObjectProperty(P)</code>	如果 xPy 且 yPz ，则 xPz
<code>owl:SymmetricObjectProperty</code>	如果 xPy ，则 yPx

图 13-6 RDF、RDF-S 和 OWL 的部分内置谓词

【例 13-15】 Turtle 符号中类构造器的例子，论元间使用空间：

```
owl:MinCardinality(2,owns,building)
```

表示拥有两个或两个以上的建筑的个体的类。也就是说，它是集合 $\{x: \exists i_1 \exists i_2 x: \text{owns } i_1 \text{ 且 } x: \text{owns } i_2 \text{ 且 } i_1 \neq i_2\}$ 。必须在声明中使用这个类构造器，例如，声明某个体是这个类的一个成员或某个类是其他类的等价类。

【例 13-16】 考虑公寓楼的亚里士多德定义。一座公寓楼可以看做是一栋居住建筑，它有多个单元，并且可以被租用。（这是相对于每个单元可以单独出售的独立产权公寓楼，或只有一个单元的房屋）假设 `ResidentialBuilding` 类是 `Building` 的一个子类。

先定义功能对象属性 `numberOfUnits`，域为 `ResidentialBuilding`，范围为 $\{\text{one, two, moreThanTwo}\}$ 。在 Turtle 中表示为：

```
:numberOfUnits rdf:type owl:FunctionalObjectProperty;  
                rdfs:domain :ResidentialBuilding;  
                rdfs:range owl:OneOf(:one :two :moreThanTwo).
```

相似的，可以定义功能对象属性 `ownership`，作用域为 `ResidentialBuilding`，值为 $\{\text{rental, ownerOccupied, coop}\}$ 。

可以定义一个公寓大楼为 `ResidentialBuilding`，其属性 `numberOfUnits` 的值为 `moreThanTwo`，属性 `ownership` 的值为 `rental`。为了使用 OWL 描述上述关系，定义一个类的属性 `numberOfUnits`，其值为 `moreThanTwo`，属性 `ownership` 的值为 `rental`，并且说 `ApartmentBuilding` 与这些类的交集等价。Turtle 的表示为：

```
:ApartmentBuilding  
  owl:EquivalentClasses  
    owl:ObjectIntersectionOf (  
      owl:ObjectHasValue(:numberOfUnits :moreThanTwo)  
      owl:ObjectHasValue(:ownership :rental)  
      :ResidentialBuilding).
```

569
}
570

这个定义可以回答关于公寓楼的问题，比如所有权和单元数目。

前面的示例并没有真正定义 ownership。系统不知道 ownership 意味着什么，而希望用户理解其含义。如果要应用本体，应该保证所使用的属性和类与其他用户使用此本体所指的是相同事物。

领域本体(domain ontology)是一个特定的领域的本体。现存的本体的适用领域都比较狭小，大多数是为特定应用而编写的。编写领域本体以便知识共享的指导方针如下：

- 尽量使用现存的本体，这意味着你的知识库将能够与使用相同本体的其他人的进行交互。
- 如果现存的本体不完全符合你的需求，尽量在已有的本体上扩展。不要重新构造本体，因为这样会导致那些想使用最好本体的人也不得不重新选择。如果你的本体包括并扩展了其他人的本体，其他人也会选择你的本体，因为它们的应用可以自由交互。
- 确保本体和邻近的本体是可以集成的。例如，关于度假村的本体必须与食品、海滩、休闲活动等本体进行交互。尽量确保它们对相同的事物使用相同的术语。
- 尽量适应更高级别的本体(见下文)。这将使它更容易与他人的知识相结合。
- 如果必须设计一个新的本体，应广泛征求其他潜在用户的意见，这将使新的本体最有用和最可能被广泛采用。
- 遵循命名约定。使用成员单数作为标识符命名一个类。例如，将类命名为 Resort 而不是 Resorts。不要因为 Resort 是一种概念而将 Resort 类称为 ResortConcept。当命名类和属性时要考虑到它们如何使用。只看做 Resort 的概念，而不是指一个具体 Resort(参见下文“类和概念”)如称“ r_1 是类 Resort”比“ r_1 是类 Resorts”更好，也比“ r_1 是类 ResortConcept”更好。
- 指定本体之间的匹配。有时当本体独立产生时必须要做本体匹配，如果能够避免匹配是最好的，它使得利用本体的知识更复杂，因为有多种方式表示同一事物。

571

使用 Turtle 描述 OWL，比使用 XML 更容易阅读。然而，OWL 是一种不方便大多数人阅读的低级语言。OWL 被设计成一种方便机器阅读的规范。有很多 OWL 的编辑器，例如 Protégé(<http://protege.stanford.edu/>)。本体编辑器应该支持如下特征：

- 应该可以提供最合理的抽象层面上输入抽象本体的方式。
- 如果用户想使用一个概念，本体编辑器应容易找到输入概念对应的术语，或者确定没有输入概念对应的术语。
- 为某类人确定术语的相应含义。
- 本体编辑器应该尽可能容易地检查本体的正确性(例如匹配用户意图的术语的解释)。
- 本体编辑器应该创建一个其他人能够使用的本体。这意味着本体编辑器应该尽可能使用标准化的语言。

类和概念

由于符号表示概念，所以很容易将类称作**概念**(concept)：从内部的表示映射到符号表示的对象或关系。

例如，由于不存在独角兽，而只有独角兽的概念，所以很容易将独角兽类称为“独角

兽概念”，然而独角兽与独角兽的概念是不同的：一个是动物，一个是知识的子类。独角兽有四条腿，头上长有一只角。独角兽概念没有腿或角。独角兽一般不会出现在本体的一个类中，而只能出现在独角兽的概念中。没有独角兽的实例，然而独角兽概念则可以有許多实例。如果希望表示独角兽这种动物，应该使用术语“独角兽”。如果希望表示独角兽概念，应该使用“独角兽概念”表示。我们不能说独角兽概念有四条腿，因为知识实例没有腿，只有动物(和家具)有腿。

另一个是关于地质构造板块的例子，这种板块可能存在了数百万年。而提出“地质构造板块概念”的时间不到一百年。人脑中可以有地质运动产生的板块这种概念，但是人脑中不能存在真实的地质运动产生的板块。很显然“地质构造板块”和“地质构造板块概念”具有不同的属性，有很大的不同。当表示“地质构造板块”时不要使用“地质构造板块概念”，反之亦然。

构建本体时常犯的错误是称对象为概念。虽然你可以自由进行命名，但是当且仅当别人接受你的本体定义时，知识共享才变得有意义。

13.3.2 顶层本体

例 13-16 中定义了可用于公寓大楼的知识库的领域本体。每个领域本体隐式或显式地假定它能够适用于更高层次的本体。建立一个一致的顶层的本体是有重要意义的，可以方便其他本体的引用和适应，对更高一级本体的适应会使得本体间更容易交互。

BFO(Basic Formal Ontology, 基本形式化本体)是一种顶层本体。BFO 的分类见图 13-7。

BFO 的最顶层为**实体(entity)**。OWL 称为层次事物的顶层。所有事物都可以称为实体。

实体包括**持存体(continuant)**和**偶然体(occurrent)**。持存体是长时间存在的实体，例如人、国家、笑脸、花的气味或电子邮件。偶然体的存在时间很短暂，如生命、一个微笑、一朵花的开放或发送电子邮件这个动作。可以通过判断是否为实体的组成部分来区分持存体和偶然体：例如手指是人的一部分，而不是生活的一部分；幼儿期是人生的一部分，而不是人的一部分。持存体参与偶然体的创建，持续一段时间的过程和瞬时发生的事件都是偶然体。

持存体可以是一个**独立持存体(independent continuant)**、一个**从属持存体(dependent continuant)**或一个**空间区域(spatial region)**。独立持存体可以是一个单独存在的实体，也可以是另一个实体的一部分，例如，人、人脸、笔、苹果的表面、赤道、国家、大气都是独立持久体。从属持存体依赖于其他实体而存在，但不是其他实体的一部分，例

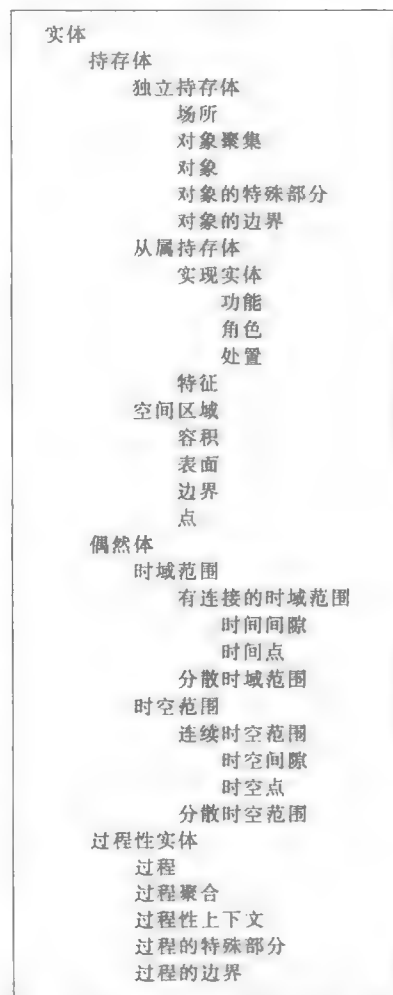


图 13-7 基本形式化本体分类。图中缩进表示子类关系，每个分类是其上面最低缩进分类的直接子类

如,一个微笑、花的气味、笑的能力只能依赖于另一个对象而存在。空间区域是空间中的一个区域,例如,一个炸圈饼所占用的空间,一个县的边界,或一个有最佳视角的景观观点。

独立持存体可以进一步划分如下:

- **场所(site)**,是由其他持存体定义的。例如,一个甜甜圈的孔、一个城市、某个人的嘴、一个房间都属于场所。场所可能会存在于另一个空间区域中随着包含它的空间而移动。
- **对象聚集(object aggregate)**,由一类对象组成,如一群羊、一支足球队或一堆沙子。
- **对象(object)**,是一个自我连接的实体,对象增加或损失一部分仍然是原对象(例如,失去了一些头发、一种信仰,甚至一条腿的人仍然是同一个人)。常见的对象有杯子、人、电子邮件、相对论或系鞋带的技巧。
- **对象的特殊部分(fiat part of an object)**,是对象没有明确边界的部分,如一个城市的危险地区、一个组织样本或一个海滩僻静的部分。
- **对象的边界(boundary of an object)**,持存体的边界,例如地球的表面、细胞壁。

空间区域可以是三维(立体)的、二维(平面)的、一维(线)的或零维(点)的。空间区域不依赖于其他对象。空间区域不随对象位置的移动而移动。

从属持存体是一个特征实体或可实现的实体。**特征(quality)**是某种类型的所有对象始终拥有的,例如,一包糖的质量、一只手的形状、杯子的脆弱性、视图的美、光的亮度、海洋的气味。虽然特征的值是可以改变的,但是糖总是有质量的,手总是有形状的。这与**实现实体(realizable entity)**形成对比,实现实体的值可以为空,可以随着时间而改变。一个可表示的实体具有下列特征之一:

- **功能(function)**,指定一个对象的功能。例如,一个咖啡杯的功能是盛放咖啡;心脏的功能是泵血。
- **角色(role)**,指定一个目标,不必是对象设计需要的,但是可实现的。例如,法官是一种角色,送咖啡是一种角色,摆放计算机显示器的桌子是一种角色。
- **处置(disposition)**,对象上发生的事情,例如,如果坠落,杯子摔破就是一个处置;如果不冷藏,蔬菜腐烂也是一个处置;如果火柴不潮,就可以点燃也是一个处置。

实体的另外一个主要类别是偶然体,下列实体都属于偶然体:

- **时域范围(temporal region)**是一个时间域。时域范围或是连续的(如果两个点在该区域,那么这两个点之间的点也在该区域)或是分散的。连续的时域范围或是间隔的或是瞬间的(时间点)。2011年3月1日,星期二,是一个时间间隔;当天下午3:31是一个时间点。从3:00到4:00间的每个星期二,是一个分散时域范围。
- **时空范围(spatio-temporal region)**是多维的。时空范围可以是分散的或是连续的。例如,人类生活所占用的空间属于时空范围。1812年加拿大和美国之间的边界,癌症肿瘤生长的区域都是时空范围。
- **过程性实体(processual entity)**是一些可以发生或突发的实体,活动实体有时间部分(也可能有空间部分),并依赖于一个持存体。例如,乔的生命中有婴儿时期、

儿童时期、青少年时期和成年后时期等部分，这些部分都依赖于持存体——乔。一个过程性实体包括如下部分：

- **过程**(process)，具有时间属性并具有明显的结束点。例如一条生命、一段假期或诊断会议。
- **过程聚合**(process aggregate)，是进程的集合，如乐队中人们的演奏或一天之中一组飞机的飞行。
- **过程的特殊部分**(fiat part of process)，无确切结束点的进程，例如一个假日里最有趣的部分或一个操作中最重要的部分。
- **过程性上下文**(processual context)，是某些偶然体的设置，例如休息是恢复活力的设置或手术是防止感染的设置。
- **过程的边界**(boundary of a process)，是一个进程的瞬时界限，例如一个机器人开始清理实验室的时刻或出生的时刻。

上述分类有助于其他本体的建立。它明确领域本体如何与上层本体结合，保证本体方便的整合。集成本体是必要的，它允许应用来引用多个知识库，每一种知识库都可能使用不同的本体。

设计顶层本体很困难。顶层本体不能满足所有用户的需求，总会出现有问题的情形。尤其，边界情况往往不好指定。但使用一个标准的顶层本体有助于本体的连接。

13.4 查询用户和其他知识来源

在 5.3.2 节的讨论中得出用户不是知识库领域的专家的结论；然而他们往往非常了解特定情况下的细节，因此可以提供一个知识源。通常情况下，用户不知道相关的知识和需要使用的词汇，因此不可能告诉系统他们知道的知识。知识获取的一方面问题是，如何最有效地从用户那里获取知识。

最简单的问题形式是 5.3.2 节中所表示的“yes-or-no”形式。通过引入变量和函数符号，可以询问用户更复杂的问题。

【例 13-17】 考虑例 12-11 中的知识库，但没有“up”或“down”的规则。假设用户可以观察到开关的位置，为了使用户能够被询问到开关的位置的问题，问题 $up(S)$ 和 $down(S)$ 是能够被询问的。下面是对于询问“ $?lit(L)$ ”(即用户被询问的目标)的一个自顶向下证明过程的一段可能的对话。用户的回复用黑体标注：

Is $up(s_2)$ true? **yes.**

Is $up(s_1)$ true? **no.**

Is $down(s_2)$ true? **no.**

Is $up(s_3)$ true? **yes.**

答案： $L = l_2$ 。

在本例中，“up”和“down”之间没有明确的相关性，所以系统对这两种情况都会询问。

上述例子中的本体很简单，我们假设用户能够理解这个问题。一般情况下，询问用户并不是那么简单，问题必须以用户能够理解的形式设定，用户必须理解问题的含义和预期的回答。

13.4.1 函数化关系

当一个关系被函数化后,即使用户没有直接给出答案,系统也可以知道询问目标的答案。当对于每个 X ,有唯一的 Y 使得 $r(X, Y)$ 为真时,关系 $r(X, Y)$ 是函数化的。如果对于特定的 X ,系统已经找到一个 Y 使得 $r(X, Y)$ 为真,则不应该再询问更多关于 Y 的问题。

【例 13-18】 在前面例子中,当用户已经告诉系统 $up(s_2)$ 为真时,问题“Is $down(s_2)$ true?”就是多余的。最好定义一个关系 $pos(Sw, pos)$,其中位置是开关的函数。告知系统 pos 是函数化的;每个开关只有一个位置。一旦系统确定了一个开关的位置,则不会再询问关于此开关位置的问题。

询问一个函数化关系的“yes-or-no”问题通常是非常低效的。不要枚举每个开关的位置,并且问是否是那个开关,最好只问一次关于开关位置的问题,而不是重复提问。如果系统询问一个人的年龄,直接问此人的年龄要好于列举所有的年龄并逐个询问“true-or-false”。

这种普遍化的问题可能不适合于非函数化的关系。例如,如果开关可以同时安装在许多位置上,那么最好对每个位置都做询问。

一个复杂的问题是关于答案的词汇。用户不清楚知识库工程师期望得到的是什么词汇。有两个方案解决这个问题:

- 系统设计师提供一个菜单条目,用户从中选择最适合的。这种方法在条目数量少或条目可以以较小的层次结构排列时才有效。
- 系统设计师提供一个较大的能够预估所有可能的答案的词典。当用户给出一个答案,答案就被映射为系统所期望的内部形式。通常假设用户使用的词汇都是普通语言或缩写形式,因此用户不希望系统理解像“giganormous”(即“very big”)这样的词汇。

上述工作属于经验问题。有关如何设计计算机与人交互的问题属于人机交互(human-computer interaction, HCI)领域的问题。

13.4.2 更普遍的问题

yes-or-no 问题和函数化关系不能涵盖查询的所有情况。可以在询问用户的查询中加入自由变量得到通用的查询形式。

【例 13-19】 对于子目标 $p(a, X, f(Z))$,可以询问用户类似这样的问题:

X 和 Z 取何值时, $p(a, X, f(Z))$ 为真?

当然应该提交用户能够理解的问题形式。

然后期望用户对 X 和 Z 给出约束,使得子目标为真,或者回答 no,表示不存在这样的 X 和 Z 实例。这符合系统提问的查询协议。

这里会出现一些如下的问题:

- 对于系统提示的新实例,用户应该给出所有值为真的实例,还是一次给出一个实例?认可一个基于知识的主要标准是它的合理性。因此以自然的、合乎逻辑的方式来提问是非常重要的。出于这个原因,当需要另一个实例时最好给出提

示。通过这种方式,系统能够在考虑下一个个体之前对个体的深度进行探测。这种方式的效果取决于知识库的结构。

578

- 何时不问问题或不再对一个问题提问?例如,对于问题“ X 取何值可使 $p(X)$ 为真?”如果用户已经给出回答的含义为 $X = f(Z)$ (例如,“对于 Z 的所有值, $p(f(Z))$ 为真”)。在这种情况下,不应该再问用户 $p(f(b))$ 是否为真或 $p(f(h(W)))$ 是否为真,而应该问关于 $p(a)$ 或 $p(X)$ 是否为真(即问一个不同的实例)。如果用户随后对问题“ X 取何值可使 $p(X)$ 为真?”的回答为 no。意思是对这个问题没有答案,而不是说对于所有的 X , $p(X)$ 都为假(因为系统已经被告知 $X = f(b)$ 是一个为真的实例)。

一般的规则是,不应该对已经做出正面回答的问题或对用户已经回答 no 的问题再做更具体的询问。

- 在证明过程中,系统遇到问题是应该立即询问,还是应该推迟询问直到有更多变量的值被确定呢?在查询中,可能会存在一些目标,无论用户给出任何答案,都将无法达到,此时最好直接寻找实现这些目标的答案而不是询问用户。在一个查询中也存在随着自由变量的求解,最终的答案将随后被确定的情况。相比于让用户列举各种可能性而偶然碰到答案,可能等待自由变量被求解后的方式会更好些。

考虑查询“ $? p(X) \wedge q(X)$ ”,其中 $p(X)$ 是可询问的。如果只有一个 q 的实例,如 $q(k)$ 为真,那么最好拖延询问 $p(X)$ 直到 X 值确定为 k 。然后系统直接询问 $p(k)$,而不是询问用户枚举所有的 p ,直到问到 $p(k)$ 为真。然而,如果 $q(X)$ 的实例是一个很大的数据库,而只有很少的 X 值使得 $p(X)$ 为真,那么最好询问 $p(X)$ 的实例,然后利用 $p(X)$ 去检查数据库,而不是对数据库的每一个元素都提出 yes-or-no 的问题。直接询问还是延迟询问,哪一种方式更好取决于对于使得 $p(X)$ 为真的个体数目的估计,取决于对于使得 $q(X)$ 为真的个体数目的估计和询问用户的相关成本。

上述实际的问题对于一个 KB 系统设计一个友好的用户界面是非常重要的。

13.5 实现基于知识的系统

对于一个 Agent 来说,能够表示自己的推理是非常重要的。Agent 表示自己的推理被称为**反射**(reflection)。明确地表示 Agent 的推理使 Agent 能够更加灵活地推理,以便于设计师可以为应用设计最合适的语言。

本节着重介绍反射的用途之一,即实现为特定应用所需特征而构建的新语言而提供的轻量级工具。通过这种轻量级工具很容易实现新的语言和其他工具,并且保证语言和工具可以随着应用的演变而演变。

579

语言的**元解释器**(meta-interpreter)是用同一语言书写的语言解释器。这种解释器很有用,在原型上的修改方便具有有用特征的新语言的快速构建。一旦一种语言证明了它的效用,这种语言的编译器就会被开发出来以提高效率。

在另一种语言内部实现一种新语言,实现的语言被称为**基语言**(base language),或者称为**对象语言**(object language),已经被实现的语言称为**元语言**(metalanguage)。在基语言中的表达式属于**基本级**(base level),在元语言中的表达式属于**元级**(meta-level)。本节首先为 12 章提出的“限定子句语言”定义一个元解释器,然后说明如何修改或扩展基语言,以及如何通过修改元解释器得到解释和调试工具。

13.5.1 基语言和元语言

我们需要对基本级表达式做出表示,使得解释器可以通过操纵基本级表达式来产生答案。起初,基语言也是定义子句的语言。在前文中已经表述过,限定子句语言是由术语、原子、主体和子句构成。

元语言是指基语言的语法元素。因此,元级符号代表基本级术语、原子、主体和子句。基本级术语表示域中被模拟的对象,并且基本级的谓词表示域中的关系。

一个元解释器在为一个新的语言编写解释器方面的优点之一是其对象级能够使用元级结构。当编写一个逻辑编程元解释器时,需要对如何表示变量做出选择。在非基表示(non-ground representation)中,基本级术语的表示与元语言中的相同,所以,特别的,基本级变量可表示为元级变量。这是相对于基础表示(ground representation)而言的,在基础表示中,基语言变量表示为元语言中的常量。非基表示意味着,元级统一被有效地用于基本级术语的统一。基础表示可以实现更复杂的统一的模型。

【例 13-20】 在非基表示中,基本级术语 $foo(X, f(b), X)$ 将被表示为元级术语 $foo(X, f(b), X)$ 。

在基础表示中,基本级术语 $foo(X, f(b), X)$ 将被表示为 $foo(var(x), f(b), var(x))$, 其中 var 是元级函数符号,代表给定名字的变量作为其参数。

我们将为限定子句生成非基表示。元语言必须能够表示所有的基本级结构。

基本级变量、常量和函数符号都可以表示为相应的元级变量、常量和函数符号。因此,所有基本级中的术语可表示为元级中相同的术语。基本级谓词符号 p 由相应的元级函数符号 p 表示。所以,基本级原子 $p(t_1, \dots, t_k)$ 被表示为元级术语 $p(t_1, \dots, t_k)$ 。

580

基本级主体也表示为元级术语。如果 e_1 和 e_2 是元级术语,代表基本级原子或主体,令元级术语 $oand(e_1, e_2)$ 代表 e_1 和 e_2 的基本级合取。因此, $oand$ 是一个元函数符号,表示基本级合取。

基本级子句可以表示为元级原子。基本级规则“ h if b ”表示为元级原子 $clause(h, b')$, 其中, b' 表示 b 的主体。基本级原子子句 a 被表示为元级原子 $clause(a, true)$, 其中元级常量 $true$ 表示基本级空主体。

【例 13-21】 例 12-11 的基本级子句

```
connected_to( $l_1$ ,  $w_0$ )
connected_to( $w_0$ ,  $w_1$ )  $\leftarrow$  up( $s_2$ )
lit( $L$ )  $\leftarrow$  light( $L$ )  $\wedge$  ok( $L$ )  $\wedge$  live( $L$ )
```

可以表示为元级事实:

```
clause(connected_to( $l_1$ ,  $w_0$ ), true)
clause(connected_to( $w_0$ ,  $w_1$ ), up( $s_2$ ))
clause(lit( $L$ ), oand(light( $L$ ), oand(ok( $L$ ), live( $L$ ))))
```

为了使基本级更具可读性,使用中缀函数符号“ $\&$ ”代替 $oand$, 则 $oand(e_1, e_2)$ 可写为 $e_1 \& e_2$ 。连接符号“ $\&$ ”是元语言的中缀函数符号,是一种基语言的原子之间的算子。这只是 $oand$ 表示的语法变化。这种中缀运算符的使用使得基本级公式更容易阅读。

使用 $h \leftarrow b$ 代替 $clause(h, b)$, 其中 \leftarrow 是中缀元级谓词符号。因此,基本级子句“ $h \leftarrow a_1 \wedge \dots \wedge a_n$ ”表示为元级原子:

```
 $h \leftarrow a_1 \& \dots \& a_n$ 
```

如果相应的基本级子句是基本级知识库的一部分，那么上述的元级原子为真。在元级中，这个原子能够像任何其他原子一样使用。

图 13-8 总结了如何使用元语言表示基语言。

语法结构		语法结构的元级表示	
变量	X	变量	X
常量	c	常量	c
函数符号	f	函数符号	f
谓词符号	p	函数符号	p
and 操作符	\wedge	函数符号	$\&$
if 操作符	\leftarrow	谓词符号	\leftarrow
子句	$h \leftarrow a_1 \wedge \cdots \wedge a_n$	原子	$h \leftarrow a_1 \& \cdots \& a_n$
子句	h	原子	$h \leftarrow true$

图 13-8 基语言的非基表示

【例 13-22】 采用中缀符号表示法将例 13-21 中的基本级子句表示为元级事实：

```
connected_to( $l_1, w_0$ )  $\leftarrow true$ 
connected_to( $w_0, w_1$ )  $\leftarrow up(s_2)$ 
lit( $L$ )  $\leftarrow light(L) \& ok(L) \& live(L)$ 
```

与例 13-21 中的表示方法相比较，上述表示方法的可读性更好，但对计算机而言，它们本质上是相同的。

元级函数符号“&”和元级谓词符号“ \leftarrow ”不是预定义的元级符号，可以使用任何其他的符号。中缀符号仅仅是为了增强易读性。

13.5.2 普通的元解释器

本节介绍一个非常简单的利用限定子句语言编写的普通的元解释器(vanilla meta-interpreter)，用于解释限定子句语言。后续的章节将增强这种元解释器以提供额外的语言构建和知识工程工具。在考虑后面提及的更复杂的元解释器之前，首先来理解一个简单的例子。

图 13-9 定义了限定子句语言的元解释器。这是一种公理化的关系 *prove*；当基本级主体 *G* 是基本级子句的一个逻辑结果时，*prove*(*G*)为真。

```
%当基级主体 G 是使用谓词符号“ $\leftarrow$ ”定义的基级子句的逻辑结果，prove(G)为真
prove(true).
prove((A & B))  $\leftarrow$ 
    prove(A)  $\wedge$ 
    prove(B).
prove(H)  $\leftarrow$ 
    (H  $\leftarrow$  B)  $\wedge$ 
    prove(B).
```

图 13-9 普通的限定子句元解释器

正如公理化任何其他关系一样，子句在预期的解释中取真值，确保子句覆盖了所有的情况，并且通过递归做了一些简化。这种元解释器本质上涵盖了在子句或查询的主体中的每种可能的情况，并且给出如何解决每种情况的方案。主体可以是空的，或是一种合取，或是一个原子。很容易证明空的基本级主体为真。可以通过证明 *A* 和 *B* 为真来证明基本

级合取 $A \& B$ 为真。为了证明原子 H 为真, 首先找到一个 H 的基本级子句作为头部, 并且证明该子句的主体为真。

【例 13-23】 考虑图 13-10 中依据例 12-11 改编的基本级知识库的元级表示。该知识库由元级原子构成, 有相同的谓词符号, 即“ \Leftarrow ”。下面介绍由普通的元解释器和 \Leftarrow 子句组成的知识库是如何自顶向下工作的。

使用下面的查询询问基本级目标 $live(w_5)$:

$?prove(live(w_5))$

$prove$ 的第三个子句是唯一符合此查询的子句。接着寻找形式为 $live(w_5) \Leftarrow B$ 的子句, 得到

$live(W) \Leftarrow connected_to(W, W_1) \& live(W_1)$

W 与 w_5 一致, B 与 $connected_to(w_5, W_1) \& live(W_1)$ 一致, 接下来尝试证明

$prove((connected_to(w_5, W_1) \& live(W_1)))$

$prove$ 的第二个子句可以证明上式, 然后证明

$prove(connected_to(w_5, W_1))$

使用 $prove$ 的第三个子句可以证明上式。寻找首部与下式首部一致的子句

$connected_to(w_5, W_1) \Leftarrow B$

找到 $connected_to(w_5, outside) \Leftarrow true$, 将 W_1 与 $outside$ 绑定, 使用 $prove$ 的第一个子句可以推导出 $prove(true)$ 。

合取的后半部分 $prove(live(W_1))$, 其中 $W_1 = outside$, 可归结为证明 $prove(true)$ 。

```
lit(L) ←
    light(L) &
    ok(L) &
    live(L).

live(W) ←
    connected_to(W, W1) &
    live(W1).

live(outside) ← true.
light(l1) ← true.
light(l2) ← true.
down(s1) ← true.
up(s2) ← true.
up(s3) ← true.
connected_to(l1, w0) ← true.
connected_to(w0, w1) ← up(s2) & ok(s2).
connected_to(w0, w2) ← down(s2) & ok(s2).
connected_to(w1, w3) ← up(s1) & ok(s1).
connected_to(w2, w3) ← down(s1) & ok(s1).
connected_to(l2, w4) ← true.
connected_to(w4, w5) ← up(s3) & ok(s3).
connected_to(p1, w5) ← true.
connected_to(w5, w5) ← ok(cb1).
connected_to(p2, w6) ← true.
connected_to(w6, w5) ← ok(cb2).
connected_to(w5, outside) ← true.
ok(X) ← true.
```

图 13-10 室内布线的基本级知识库

13.5.3 扩展基语言

可以通过修改元解释器来改变基语言。添加子句可以增加语言的证明能力。向元解释器的规则中增加条件可以限制语言的证明能力。

在所有的实用系统中, 不是每一个谓词都可以用子句定义的。例如, 在当前可以做快速算术运算的机器上实现计算公理化是不切实际的。最好的方法是直接调用底层系统, 而不是公理化谓词。假设谓词 $call(G)$ 可以直接求 G 的值, $call(p(X))$ 与 $p(X)$ 是一致的。由于限定子句语言不允许使用自由变量作为原子, 所以需要加上谓词 $call$ 。

内置的程序可以通过定义元级关系 $built_in(X)$ 在基本级上被计算, 如果 X 的所有实例被直接计算, 那么 $built_in(X)$ 为真; 其中 X 必须是表示基本级原子的元级变量。不要假设 $built_in$ 可以作为内置关系被提供。 $built_in$ 可以像任何其他关系一样被公理化。

基语言被扩展后可以用于在子句的主体中的析取(或者包含)。

当 A 在解释 I 中为真, 或当 B 在解释 I 中为真(或者 A 和 B 都在解释 I 中为真)时, 析取(disjunction) $A \vee B$ 为真。

如果允许在基本级的主体中进行析取, 则不需要在元语言中析取。

【例 13-24】 基本级规则的一个例子

$can_see \Leftarrow eyes_open \& (lit(l_1) \vee lit(l_2))$

582

583
7
584

表示如果 *eye_open* 为真并且 *lit*(*l*₁) 或者 *lit*(*l*₂) 为真(或两者都为真), 那么 *can_see* 为真。◀

图 13-11 说明元解释器允许对内置程序直接求值, 并且允许对规则主体析取。实现上述功能需要一个内置的断言库, 并且假设 *call*(*G*) 是可以在元级上证明 *G* 的一种方式。

对于这种解释器, 元级和基本级是不同的语言。基本级允许在主体内析取。元级不需要为基语言提供析取功能。元级需要一种方式来解释 *call*(*G*), 而基本级则不需要。然而, 可以在基语言中添加下面的元级子句来解释命令 *call*(*G*):

```
prove(call(G))←
    prove(G)
```

```
%如果基级主体 G 是基级知识库的逻辑结果, 则 prove(G) 为真
prove(true).
prove((A&B))←
    prove(A) ∧
    prove(B).
prove((A ∨ B))←
    prove(A).
prove((A ∨ B))←
    prove(B).
prove(H)←
    built in(H) ∧
    call(H).
prove(H)←
    (H←B) ∧
    prove(B).
```

图 13-11 使用内置调用 *call* 和析取的元解释器示例

13.5.4 深度有限搜索

上一节展示了如何添加额外的元级子句来扩展基语言。本节说明如何向元级子句添加额外的条件来缩减基语言的范围。

有用的元解释器是一种可以实现深度有限搜索的解释器。它可用于查找简短的证明或者可以作为迭代深度搜索的一部分, 在每个阶段执行增加限定的重复的深度限定、深度优先搜索。

图 13-12 给出了一个公理化的关系 *bprove*(*G*, *D*), 如果 *G* 可以利用深度小于或等于非负整数 *D* 的证明树得到证明, 那么关系 *bprove*(*G*, *D*) 为真。此图使用 Prolog 的中缀谓词符号 *is*, 其中如果 *V* 是表达式 *E* 的数值, 则“*V is E*”为真, 在这个表达式中“*-*”是中缀减法函数符号。因此, 如果 *D*₁ 比数值 *D* 小 1, 则“*D*₁ *is* *D-1*”为真。

```
%如果 G 可以被一棵深度小于或等于数值 D 的证明树证明, 那么 bprove(G, D) 为真
bprove(true, D).
bprove((A & B), D)←
    bprove(A, D) ∧
    bprove(B, D).
bprove(H, D)←
    D ≥ 0 ∧
    D1 is D-1 ∧
    (H←B) ∧
    bprove(B, D1).
```

图 13-12 元解释器实现深度有限搜索

如果在查询中 D 是固定值, 元解释器就不会陷入死循环, 但是此元解释器不能证明其深度大于 D 。因此将 D 设为固定值的这个元解释器是不完整的。然而, 如果 D 足够大, 对于 prove 元解释器的每一个证明都可以由这个元解释器找到。迭代深度搜索的实质是利用执行重复的深度有限搜索, 每次增加深度的限定来实现的。有时深度有限元解释器可以找到 prove 不能找到的证明。在 prove 元解释器探求到所有证明之前就陷入了无穷循环时会发生这种情况。

这不是构建有界元解释器唯一的方式。深度有限元解释器还可以通过将其他指标替代深度。例如, 使用树的节点数替代树的最大深度。还可以通过改变第二规则使合取产生成本(参见习题 13.7)。

586

13.5.5 元解释器构建证明树

为了实现 5.3.3 节中的 how 命令, 解释器通过证明树的表示方式来导出答案。元解释器可以被扩展来建立一个证明树。图 13-13 给出一个实现内置谓词并建立证明树的元解释器。通过遍历这棵证明树可以实现“how”询问。

```
%当基级主体 G 是基级知识库的逻辑结果, 并且 T 表示相应证明过程的证明树, 则 hprove(G, T)为真
hprove(true, true).
hprove((A & B), (L & R))←
    hprove(A, L) ∧
    hprove(B, R).
hprove(H, if(H, built_in))←
    built_in(H) ∧
    call(H).
hprove(H, if(H, T))←
    (H←B) ∧
    hprove(B, T).
```

图 13-13 元解释器建立一个证明树的过程

587

【例 13-25】 考虑例 12-11 中接线域的基本级子句和基本级查询“ $?lit(L)$ ”。元查询“ $?hprove(lit(L), T)$ ”的答案是 $L=l_2$, 其证明树 T 如下:

```
T = if (lit(l2),
    if (light(l2), true) &
    if (ok(l2), true) &
    if (live(l2),
        if (connected_to(l2, w4), true) &
        if (live(w4),
            if (connected_to(w4, w3),
                if (up(s3), true)) &
            if (live(w3),
                if (connected_to(w3, w5),
                    if (ok(cb1), true)) &
                if (live(w5),
                    if (connected_to(w5, outside), true) &
                    if (live(outside), true)))))))).
```

尽管这棵树如果被适当地格式化后能够被理解, 但仍然对用户有技术上的要求, 才能

理解这棵树。5.3.3节中的“how”询问遍历了这棵树。用户只需要看到子句，而不需要看这棵树。参见习题13.12。

13.5.6 可询问用户的元解释器

图13-14给出一个可询问用户的解释器的伪代码。这个解释器假设有外部数据库记录查询的答案，当查询被回答时更新数据库。元级形式 $answered(H, Ans)$ 被添加。如果向用户询问 H ，用户给出的回答 Ans 或为 yes 或为 no ，那么 $asked(H, Ans)$ 都为真。 $unanswered(H, Ans)$ 意味着对于任何的 Ans ， $answered(H, Ans)$ 在数据库中都没有真值。注意第四子句的字面意思是当答案为 yes 时，该子句生效，但无论用户回答 yes 还是 no 答案都会被记录下来。

图13-15给出了可以用来查找 why 问题的原始规则列表的元解释器。 $wprove$ 的第二个参数是在证明树的当前部分中每个规则的头部形如 $(H \Leftarrow B)$ 的子句列表。这个元解释器可以与图13-14中做询问的元解释器相结合。当用户使用 why 回答问题时，上述子句列表可提供产生当前子目标的规则集。5.3.3节中的 why 查询可以通过遍历子句列表和向上遍历证明树，一次一步来实现。

```
%当基级主体 G 是基级知识库的逻辑结果，并且用户已经回答 yes/no 查询，则  $aprove(G)$  为真
 $aprove(true).$ 
 $aprove((A \& B)) \leftarrow$ 
     $aprove(A) \wedge$ 
     $aprove(B).$ 
 $aprove(H) \leftarrow$ 
     $askable(H) \wedge$ 
     $answered(H, yes).$ 
 $aprove(H) \leftarrow$ 
     $askable(H) \wedge$ 
     $unanswered(H) \wedge$ 
     $ask(H, Ans) \wedge$ 
     $record(answered(H, Ans)) \wedge$ 
     $Ans = yes.$ 
 $aprove(H) \leftarrow$ 
     $(H \Leftarrow B) \wedge$ 
     $aprove(B).$ 
```

图 13-14 可询问用户的元解释器的伪代码

```
%如果基级主体 G 是基级知识库的逻辑结果，并且 A 是 G 在原始查询证明树中的初始规则列表，
则  $wprove(G, A)$  为真
 $wprove(true, Anc).$ 
 $wprove((A \& B), Anc) \leftarrow$ 
     $wprove(A, Anc) \wedge$ 
     $wprove(B, Anc).$ 
 $wprove(H, Anc) \leftarrow$ 
     $(H \Leftarrow B) \wedge$ 
     $wprove(B, [(H \Leftarrow B) | Anc]).$ 
```

图 13-15 元解释器收集 why 问题的原始规则

13.5.7 推迟目标

推迟(delay)目标是元解释器的最有用的功能之一。元解释器将某些目标收集到一个列表中,而不是去证明它们。在证明的最后阶段,系统导出结论,如果推迟的目标都取真值,则结果为真。

收集可推迟目标的原因如下:

- 为了实现基于一致性的诊断和最佳推导;
- 等待后续调用给出变量的值而推迟带有变量的子目标;
- 为了省去创造新规则的中间步骤——例如,如果用户或数据库询问推迟的目标。这被称为**部分评价**(partial evaluation),用于基于解释的学习过程,以省去证明的中间步骤。

图 13-16 给出了一个提供推迟的元解释器。使用元级事实 $delay(G)$ 表示可以推迟基本级原子 G 。可推迟的原子能够被收集到一个没有被证明的列表中。

假设能够证明 $dprove(G, [], D)$ 。令 D' 表示可延迟目标 D 的列表中的基本级原子的合取,则 $G \leftarrow D'$ 是上述子句的一个逻辑结果,并且对于所有的 $d \in D$, $delay(d)$ 为真。

```
%如果  $D_0$  出现在  $D_1$  的尾部,并且  $G$  是  $D_1$  中可延迟的原子的合取,则  $dprove(G, D_0, D_1)$  为真
dprove(true, D, D).
dprove((A & B), D1, D2) ←
    dprove(A, D1, D2) ∧
    dprove(B, D2, D1).
dprove(G, D, [G | D]) ←
    delay(G).
dprove(H, D1, D2) ←
    (H ← B) ∧
    dprove(B, D1, D2).
```

图 13-16 收集延迟目标的元解释器

【例 13-26】 作为基于一致性的诊断推迟的例子,考虑如图 13-10 所示的无 ok 规则的基本级知识库。假设 $ok(G)$ 是可推迟的,将其表示为如下元级事实:

```
delay(ok(G))
```

查询

```
ask dprove(live( $p_1$ ), [], D)
```

的答案为 $D = [ok(cb_1)]$ 。如果 $ok(cb_1)$ 为真,那么 $live(p_1)$ 为真。

查询

```
ask dprove((lit( $l_2$ ) & live( $p_1$ )), [], D)
```

的答案为 $D = [ok(cb_1), ok(cb_1), ok(s_3)]$ 。

如果 cb_1 和 s_3 属于 ok , 那么 l_2 是亮的并且 p_1 是活动的。

注意, $ok(cb_1)$ 在列表中出现两次。 $dprove$ 不检查列表中是否存在多个可推迟的实例,一个 $dprove$ 的成熟版本不会增加重复的推迟。参见习题 13.8。

13.6 本章小结

- “个体-特征-值”三元组形式便于灵活通用地表示关系。
- 本体是知识共享的基础。

- OWL 本体由个体、类和属性构建。类是一个实际和潜在个体的集合。属性将数据类型或个体联系起来。
- 根据语义内容进行解释推理和调试知识的能力能够改善基于知识的系统的可用性。
- 元解释器可以实现一个可定制的适合表示语言需要的轻量级知识库系统。

13.7 参考文献及进一步阅读

Brachman 和 Levesque [2004] 概述了如何表示知识的方法。Davis [1990] 介绍了常识性推理中丰富的知识表示问题。Brachman 和 Levesque [1985] 发表了很多经典的知识表示的论文。Woods [2007] 给出有关语义网的最新概述。

591

如果希望从哲学和计算机的角度了解本体, 可参阅 Smith [2003]。语义 Web 的概述参阅 Antoniou 和 Van Harmelen [2008]; Berners-Lee、Hendler 和 Lassila [2001]; Hendler、Berners-Lee 和 Miller [2002]。本章的 OWL 描述基于 OWL-2; 有关 OWL-2 的描述可以参阅 Hitzler、Krotzsch、Patel-Schneider 和 Rudolph [2009]; Motik、Patel-Schneider 和 Parsia [2009b]; Motik、Patel-Schneider 和 Grau [2009a]。Turtle 是 Beckett 和 Berners-Lee [2008] 提出的一种语言。

Grenon 和 Smith [2004] 描述了 BFO。其他顶层本体, 包括 DOLCE [Gangemi, Guarino, Masolo, Oltramari, 2003], SUMO [Niles, Pease, 2001], Cyc [Panton, Matuszek, Lenat, Schneider, Witbrock, Siegel, Shepard, 2006]。Noy 和 Hafner [1997] 比较了不同的顶层本体。

Bowen [1985] 和 Kowalski [1979] 讨论了元解释器逻辑。Abramson 和 Rogers [1989] 总结了现有的元解释器。

13.8 习题

13.1 本练习的目的是探讨相同域的多重表示。

在一个 3×3 方格上的 tic-tac-toe 游戏中, 两名玩家 X 和 O, 交替地在空白位置留下标记。当轮到 X 放置标志时, 如果他可以将三个 X 标志在同一条直线上, 那么 X 获胜。例如, 在如下状态中, 在左侧中间的空位置上留下 X 标记, X 便可以获胜。

X	O	O
	X	
X		O

Fred、Jane、Harold 和 Jennifer 每个人编写一个程序使得在已知一个游戏的状态下, X 在下一步可获胜。他们每个人要决定游戏状态的不同表示, 比较他们的表示方法。

Fred 使用一个有三行的列表表示此游戏的状态, 列表的每行都包含三个元素, 这三个元素分别是 x 、 o 或 b (表示空白)。Fred 用如下的列表表示上面的状态:

$[[x, o, o], [b, x, b], [x, b, o]]$

Jane 使用两个数字, 即用纵横坐标描述每个点的位置。例如, 将左上角的 X 的位置表示为 $pos(1, 3)$, 左下角的 X 的位置 $pos(1, 1)$, 等等。Jane 将游戏的状态表示为 $ttt(XPs, OPs)$, 这里 XPs 是 X 的位置, OPs 是 O 的位置。因此, Jane 将上述状态表示为:

$ttt([pos(1, 3), pos(2, 2), pos(1, 1)], [pos(2, 3), pos(3, 3), pos(3, 1)])$

Harold 和 Jennifer 都意识到, tic-tac-toe 游戏的位置可以用魔方来表示:

6	7	2
1	5	9
8	3	4

将游戏转化为一个两人轮流选择一个数字的游戏。同一个数字不可以被选择两次, 首先选择出总和为 15 的三个数字的玩家获胜。

Harold 将游戏状态表示为由九个元素组成的列表, 其中每个元素是 x 、 o 或 b , 这取决于魔方

592

对应的位置是由 X 控制, 还是由 O 控制, 或是空白的。因此, Harold 使用下面的列表表示上面的游戏状态:

$[b, o, b, o, x, x, o, x, b]$

Jennifer 将游戏表示为一个由 X 和 O 选择的数字列表对。她将以上游戏表示为:

$magic([6, 5, 8], [7, 2, 4])$

(a) 基于上述四种表示方法中的每一种, 写一个关系来决定 X 是否获胜, 例如 x_won_fred (State), 表示基于 Fred 的方法, 在状态 State 上 X 赢得了比赛(类似地写出其他三种表示方法)。

(b) 哪种表示更容易使用? 解释为什么。

(c) 哪种表示可以更有效地获胜?

(d) 如果某玩家获胜, 哪种表示可以以最简单的算法来确保玩家不会在下一步失败?

(e) 你认为哪种表示最好? 解释为什么, 提出一种更好的表示。

13.2 Sam 提出任何 n 元关系 $P(X_1, X_2, X_3, \dots, X_n)$ 可以表示为 $n-1$ 个二元关系, 即

$P_1(X_1, X_2)$

$P_2(X_2, X_3)$

$P_3(X_3, X_4)$

...

$P_{n-1}(X_{n-1}, X_n)$

向 Sam 解释这种方法的缺陷。如果 Sam 这样表示, 会遇到什么问题?

13.3 为整理你桌子的机器人生成一个本体, 表示经常摆放在你办公桌上的物品。考虑以下问题: (a) 机器人能感知对象, (b) 应该能够区分任务。

13.4 使用 OWL 写出海滨度假村的定义。“海滨度假村”是一个靠近海或者湖的, 可以提供游泳、住宿和饮食的地方。

13.5 豪华饭店有多个房间可供租住, 每个房间都很舒适并可以看风景, 并且至少有一间餐厅可以同时为素食者和肉食者提供饮食服务。

(a) 使用 OWL 定义豪华饭店, 做合理的假设, 规范是模糊的。

(b) 再提出豪华饭店的三个其他属性, 并对每个属性分别给出自然语言定义和 OWL 的描述。

13.6 用 BFO 对下述子句分类:

(a) 你的皮肤

(b) 前面句子结束处的句号

(c) 一个孩子在假期前的兴奋

(d) 假期回家的旅行

(e) 计算机程序

(f) 暑假

(g) 一个电话铃声

(h) 你桌上的灰尘

(i) 打扫你办公室的任务

(j) 某人是否得流感的诊断

在上述经验的基础上, 提出并证明一种 BFO 的修改方案, 考虑在 BFO 中如何区分没有覆盖或非独占的情况。

13.7 修改图 13-12 的深度有限元解释器, 使得:

(a) 界限是证明的总长度, 这里长度为出现在证明中的基本级原子实例总数。

(b) 不同的基本级原子在界限上的成本是不同的。例如, 大多数原子是零成本, 一些原子的成本为正值。

讨论为什么上述情况比使用树的深度更好或更坏?

当给定一个正的边界值, 在什么条件下会保证证明过程不会陷入死循环?

13.8 图 13-16 的程序允许复制推迟的目标。写一个 *dprove* 版本, 该版本以最简的形式返回推迟目标的最小集合。

13.9 写一个可以寻求多种信息源的元解释器。假设每个信息源由一个通用资源标识符 (URI) 识别。假定存在如下谓词:

- 如果由 *URI* 表示的资源能够给出与问题 *Q* 一致的答案, 则 *can_answer(Q, URI)* 为真。
- 如果 *R* 是 *URI* 的某种可靠性的数值度量, 则 *reliability(URI, R)* 为真。假设 *R* 的范围是 $[-100, 100]$, 其中数值越高, 意味着越可靠。
- 向 *URI* 代表的信息源询问一个问题 *Q*, 其给出的 *Answer* 是 $\{yes, no, unknown\}$ 三者之一, 则 *askSite(URI, Q, Answer)* 为真。

写一个可以利用多种信息源并返回可靠性答案的元解释器, 答案的可靠性是该信息来源使用可靠性的最低值。当没有外部来源时, 必须有一些约定 (例如, 可靠性取 200)。只可以问信息源可以回答的已经记录的问题。

13.10 写一个可以询问用户 yes-or-no 问题的元解释器, 要求该元解释器不问已知答案的问题。

13.11 扩展可询问用户的元解释器。解释谓词是函数化的并可以做出相应的反应。

13.12 元解释器构建了图 13-13 中的证明树, 写一个程序融合该元解释器产生的树, 使用 how 问题遍历此树。

13.13 写一个支持 how 和 why 查询的元解释器。允许在 why 查询后增加 how 查询。解释这个程序如何起作用。

13.14 写一个支持迭代深度搜索的限定子句的元解释器。要求该元解释器与深度优先搜索算法返回的查询结果一致。

13.15 建立一个迭代深度回溯推理系统, 找到最小的一致性的可假设集合来蕴含一个目标。可以参考图 13-12 所示的深度有限元解释器和图 13-16 所示的推迟元解释器。深度范围应根据证明中使用的假设数来确定。假设可假设集都是研究的范围。

分为两部分完成:

(a) 基于迭代加深可假设数的方法找到蕴含某个 *g* 的最小可假设集。当 *g* 为假时, 程序会找到最小的冲突。

(b) 根据 (a), 通过交叉发现冲突, 找到 *g* 的最小解释, 得到蕴含 *g* 的最小可假设集。

13.16 为参数化的逻辑程序写一个元解释器。这些逻辑程序能够在算术表达式中使用常数, 常数值作为元解释器的输入部分。

假设环境为 *val(Parm, Val)* 的术语列表, 其中 *Val* 的值与参数 *Parm* 有关。假设输入中每个参数在环境中只出现一次。例如 $[val(a, 7), val(b, 5)]$ 。

在 AILog 中, 可以使用 “ \leq ” 作为基本级蕴含, 使用 “ $\&$ ” 作为基本级合取。AILog 将 “ \leq ” 定义为一个中缀运算符, *number* 是一个内置谓词。

(a) 如果参数 *Parm* 在环境 *Env* 中的值为 *Val*, 谓词 *lookup(Parm, Val, Env)* 为真。

(b) 如果参数化的算术表达式 *Exp* 在 *Env* 中值为 *Val*, 谓词 *eval(Exp, Val, Env)* 为真, 表达式可以是如下之一:

- 形式为 $(E_1 + E_2)$, $(E_1 * E_2)$, (E_1 / E_2) , $(E_1 - E_2)$, 其中 E_1 和 E_2 为参数化的算术表达式;
- 一个数;
- 一个参数。

假设运算符使用它们通常的含义, 数字评价自身, 参数评价环境中的值。可以使用 AILog 谓词 *is*。当非参数化表达式 *E* 的值为 *N* 时, *N is E* 为真。如果 *E* 是一个数字, *number(E)* 为真。

(c) 如果目标 *G* 是基本级知识库的逻辑结果, 谓词 *pprove(G, Env)* 为真, 其中参数在环境 *Env* 中解释。

与 AILog 交互的例子如下:

```
ailog: tell f(X,Y) <= Y is 2*a+b*X.  
ailog: ask pprove(f(3,Z),[val(a,7),val(b,5)]).  
Answer: pprove(f(3,29),[val(a,7),val(b,5)]).  
[ok,more,how,help]: ok.  
ailog: ask pprove(f(3,Z),[val(a,5),val(b,7)]).  
Answer: pprove(f(3,31),[val(a,5),val(b,7)]).  
[ok,more,how,help]: ok.  
ailog: tell dsp(X,Y) <= Z is X*X*a & Y is Z*Z*b.  
ailog: ask pprove(dsp(3,Z),[val(a,7),val(b,5)]).  
Answer: pprove(dsp(3,19845),[val(a,7),val(b,5)]).  
[ok,more,how,help]: ok.  
ailog: ask pprove(dsp(3,Z),[val(a,5),val(b,7)]).  
Answer: pprove(dsp(3,14175),[val(a,5),val(b,7)]).  
[ok,more,how,help]: ok.
```

关系规划、学习和概率推理

现在需要做的是最大可能地发展数学逻辑，从而能够完整表达关系的重要性，并将一个新的哲学逻辑置于这坚实的基础之上。这个新建立的哲学逻辑可望从这个数学基础借鉴一些精确性和确定性。如果这个任务能够实现，则我们有充分的理由去期待不久的将来会是一个与数学原理之于近代那样伟大的纯理论哲学时代。伟大的胜利孕育巨大的希望；在我们这一代，纯理论思维可能产生的伟大的结果，有可能将我们的时代与最伟大的古希腊时代比肩。

——Bertrand Russell[1917]

表示方案维度有作为顶层的个体与关系推理。关系推理允许使用压缩的表示方法，该方法可在 Agent 遇到特定个体之前建立。当一个 Agent 被引荐给一个新个体时，它能推断其与该个体的关系。本章概述在三个 AI 领域中如何扩展基于特征的表示以处理个体与关系。在其中的每一个领域，关系的表示受益于能够在知道个体之前（因此，也是在知道特征之前）建立。正如 Russell 在上面的引文中所指出的那样，关系的推理相比命题的（及基于特征的）表示具有很大的优势。

597

14.1 规划个体与关系

Agent 的目标及其环境经常使用个体和关系来描述。在 Agent 的知识库建立后，且知道其应该推理的对象之前，它需要一个独立于个体的表示法。因此，它必须采用超越基于特征的表示法。当个体已知时，可以不使用基于特征的表示法。使用不依赖于特定领域的表示法进行推理常常是有用的。

通过关系的表示，我们可以具体化时间。时间(time)可以使用指示了时间点的个体或表示了时间段的个体来表示。本节介绍两种关系的表示法，其区别在于具体化时间的方法不同。

14.1.1 情景演算

情景演算(situation calculus)背后的思想是那些(可达的)状态可由为达到它们所需要的行动来定义。这些可达到的状态被称为情景。一个情景中的真可由此情景中的关系定义为一个参数。可将情景演算视为动作的基于特征表示的关系版本。

这里，我们仅考虑一个 Agent、一个完全可观察的环境及确定性的动作。

情景演算由情景来定义。一个**情景**(situation)是下面两者之一：

- *init*：初始情景。
- *do*(*A*, *S*)：在情景 *S* 中执行动作 *A* 而产生的结果情景(假设在情景 *S* 中可以执行动作 *A*)。

【例 14-1】 考虑图 3-1 的领域背景。假设在初始状态 *init* 时，机器人 *rob* 在位置 *o109*，且邮件收发室(*mail*)中有一把钥匙 *k1*，在储藏室(*storage*)中有一个包裹。

```
do(move(rob,o109,o103),init)
```

是机器人 *rob* 由情景 *init* 的位置 *o109* 移动到位置 *o103* 后的情景。在这个情景中, *rob* 在 *o103* 位置, 钥匙 *k1* 仍然在 *mail* 位置, 包裹仍然在 *storage* 位置。

情景

```
do(move(rob,o103,mail),
  do(move(rob,o109,o103),
    init))
```

是机器人已经从位置 *o109* 移动到 *o103*, 并继续移动到 *mail*, 其当前位置在邮件收发室。假设 *rob* 随后拿起了钥匙 *k1*, 则得到的情景是:

```
do(pickup(rob,k1),
  do(move(rob,o103,mail),
    do(move(rob,o109,o103),
      init)))
```

598

在这个情景中, *rob* 正在位置 *mail* 拿着钥匙 *k1*。

一个情景可以和一个状态相联系。情景和状态有两个主要的区别:

- 如果多个动作序列最终结束于同一个状态, 那么多个情景就可能指示着同一个状态。也就是说, 情景之间的等价与状态之间的等价并不相同。
- 不是所有的状态都有相对应的情景。如果存在一个行动序列使得从初始状态出发可以到达一个状态, 则称该状态是可达的(reachable)。不可达的状态没有相对应的情景。

一些 *do(A, S)* 项并不对应任何状态。然而, 有时 Agent 必须在不知道 *A* 是否可能处于状态 *S* 或状态 *S* 是否可能达到的情况下对这种(潜在的)情景进行推理。

【例 14-2】 项 *do(unlock(rob, door1), init)* 根本不表示任何一个状态, 因为当 *rob* 不在门旁边且没有钥匙时, 是不可能打开门的。

静态(static)关系是指其真值不依赖于情景的关系, 即其真值并不随时间而改变。**动态**(dynamic)关系是指其真值依赖于情景的关系。为表示一个情景中什么是真的, 表示动态关系的谓词要有一个情景参数, 从而其真值可依赖于情景。带有情景参数的谓词被称为**流**(fluent)。

【例 14-3】 当对象 *O* 在情景 *S* 中处于位置 *L* 时, 关系 *at(O, L, S)* 为真。因此, *at* 是一个流。

原子

```
at(rob,o109,init)
```

为真, 如果机器人 *rob* 在初始情景中处于位置 *o109*。原子

```
at(rob,o103,do(move(rob,o109,o103),init))
```

为真, 如果机器人 *rob* 从初始情景开始从位置 *o109* 移动到位置 *o103*, 并且机器人在最后产生的情景中位于 *o103*。原子

```
at(k1,mail,do(move(rob,o109,o103),init))
```

为真, 如果机器人 *rob* 从初始情景开始从位置 *o109* 移动到位置 *o103*, 并且在最后产生的情景中, *k1* 位于 *mail* 处。

可通过指定情景来公理化动态关系(在指定的情景中, 动态关系的值为真)。通常, 这可通过情景的结构归纳完成。

599

- *init* 作为情景参数的公理用来指明初始情景中什么为真。

- 原始(primitive)关系通过在形如 $do(A, S)$ 的情景中, 根据情景 S 中什么为真来确定关系什么时候为真来定义。也就是说, 原始关系依据在前一个情景中什么为真来定义。
- 导出(derived)关系通过带有情景参数的子句来定义。导出关系在一个情景中是否为真依赖于在该情景中其他因素是否为真。
- 静态关系的定义跟情景无关。

【例 14-4】 假设传送机器人 *rob* 处于图 3-1 所描绘的域中。*rob* 在位置 *o109*, 包裹(*parcel*) 在储藏室(*storage*), 钥匙(*key*) 在邮件收发室(*mail*)。下面的公理描述了这个初始的情景:

$at(rob, o109, init)$

$at(parcel, storage, init)$

$at(k1, mail, init)$

邻接关系 *adjacent* 是一个动态的导出关系, 其定义如下:

$adjacent(o109, o103, S)$

$adjacent(o103, o109, S)$

$adjacent(o109, storage, S)$

$adjacent(storage, o109, S)$

$adjacent(o109, o111, S)$

$adjacent(o111, o109, S)$

$adjacent(o103, mail, S)$

$adjacent(mail, o103, S)$

$adjacent(lab2, o109, S)$

$adjacent(P_1, P_2, S) \leftarrow$
 $between(Door, P_1, P_2) \wedge$
 $unlocked(Door, S)$

注意自由变量 S , 这些子句对于所有的情景都为真。我们不能省略 S , 因为房间的邻接关系依赖于门是否打开了。这个关系在不同的情景下可能不同。

关系 *between* 是静态的, 不需要一个情景变量:

$between(door1, o103, lab2)$

我们还可以区分 Agent 是否正在搬运。如果一个对象没有被搬运, 那么我们说这个对象正处在它的位置上。我们之所以区分搬运事件是因为一个正在被搬运的对象是随着搬运它的对象而移动的。对象处于一个位置有两种情况: 它自己本就处于那个位置, 或者正被另一个处于那个位置的物体搬运。因此, 关系 *at* 是一个导出关系:

$at(Ob, P, S) \leftarrow$

$sitting_at(Ob, P, S)$

$at(Ob, P, S) \leftarrow$

$carrying(Ob1, Ob, S) \wedge$

$at(Ob1, P, S)$

注意, 这个定义允许机器人搬运一个袋子, 而这个袋子正搬运着一本书。 ◀

动作的前件(precondition)具体说明了什么时候才可能执行这个动作。当动作 A 在情景 S 中可能被执行时, 关系 $poss(A, S)$ 为真。通常, 这是一个导出关系。

【例 14-5】 Agent 总能放下它正搬运的物体:

$poss(putdown(Ag, Obj), S) \leftarrow$

$carrying(Ag, Obj, S)$

对于动作 *move*，一个自治 Agent 可以从它当前的位置移动到一个邻接的位置：

```
poss(move(Ag, P1, P2), S) ←
    autonomous(Ag) ∧
    adjacent(P1, P2, S) ∧
    sitting_at(Ag, P1, S)
```

开锁动作(*unlock*)的前件比较复杂。Agent 必须在门的正确的一侧并且拿着合适的钥匙：

```
poss(unlock(Ag, Door), S) ←
    autonomous(Ag) ∧
    between(Door, P1, P2) ∧
    at(Ag, P1, S) ∧
    opens(Key, Door) ∧
    carrying(Ag, Key, S)
```

我们并不假定 *between* 关系是对称的。一些门只能单向通行。

我们根据前一情景及转换到当前情景时发生的动作来递归地定义每个情景中的真。正如动作的基于特征的表示中使用的规则那样，**因果规则**(causal rule)明确一个关系什么时候为真，**框架规则**(frame rule)说明一个关系什么时候保持真。

601

【例 14-6】 原始的 *unlocked* 关系可通过具体说明不同的动作如何影响其为真来定义。只要开锁的动作是可执行的，那么在执行了开锁动作之后的情景中，门是打开的。这可用下面的因果规则来表示：

```
unlocked(Door, do(unlock(Ag, Door), S)) ←
    poss(unlock(Ag, Door), S)
```

假设唯一可以使门锁起来的行为是锁门的动作。因此，*unlocked* 在如下三种执行了一个动作后所产生的情景中为真：*unlocked* 在执行动作之前为真；执行的动作不是锁门的动作 *lock*；能够执行该动作。

```
unlocked(Door, do(A, S)) ←
    unlocked(Door, S) ∧
    A ≠ lock(Door) ∧
    poss(A, S)
```

这是一个框架规则。

【例 14-7】 谓词 *carrying* 可定义如下。

Agent 在拿起(*pickup*)一个对象之后就是正在搬运(*carrying*)这个对象：

```
carrying(Ag, Obj, do(pickup(Ag, Obj), S)) ←
    poss(pickup(Ag, Obj), S)
```

撤销 *carrying* 谓词的唯一动作是放下(*putdown*)动作。因此，在执行了一个动作之后，*carrying* 为真的情况有如下两种：*carrying* 在该行为之前为真；该动作不是放下对象的动作。这可由框架规则表示如下：

```
carrying(Ag, Obj, do(A, S)) ←
    carrying(Ag, Obj, S) ∧
    poss(A, S) ∧
    A ≠ putdown(Ag, Obj)
```

【例 14-8】 原子 *sitting_at*(Obj, Pos, S₁) 在由对象 Obj 移动到 Pos 而生成的情景 S₁ 中为真，只要上述动作能够执行：

```

sitting_at(Obj, Pos, do(move(Obj, Pos0, Pos), S)) ←
    poss(move(Obj, Pos0, Pos), S)

```

另一个使 *sitting_at* 为真的情况是执行 *putdown* 动作。对象位于 Agent 将它放下时所处的位置：

```

sitting_at(Obj, Pos, do(putdown(Ag, Obj), S)) ←
    poss(putdown(Ag, Obj), S) ∧
    at(Ag, Pos, S)

```

602

其他唯一可使 *sitting_at* 在一个(非初始的)情景中为真的情况是它在之前的情景中为真且没有被一个动作改变其状态。可以改变 *sitting_at* 的动作只有 *move* 和 *pickup*。这可用如下的框架公理来表示：

```

sitting_at(Obj, Pos, do(A, S)) ←
    poss(A, S) ∧
    sitting_at(Obj, Pos, S) ∧
    ∀ Pos1 A ≠ move(Obj, Pos, Pos1) ∧
    ∀ Ag A ≠ pickup(Ag, Obj)

```

请注意，上面规则中的量词不是描述规则的标准量词。也可以通过使用否定即失败推理来表示该规则：

```

sitting_at(Obj, Pos, do(A, S)) ←
    poss(A, S) ∧
    sitting_at(Obj, Pos, S) ∧
    ~move_action(A, Obj, Pos) ∧
    ~pickup_action(A, Obj)
move_action(move(Obj, Pos, Pos1), Obj, Pos)
pickup_action(pickup(Ag, Obj), Obj)

```

在上面的否定范围内，这些子句中没自由变量。

【例 14-9】 情景演算可以表示更为复杂的动作，而不只是可以通过在状态描述中简单增加和删除命题就可表示的动作。

考虑 *drop_everything* 动作。执行该动作后，Agent 会放下其正在搬运的所有东西。在情景演算中，可在 *sitting_at* 的定义中加入下面的公理来表示 Agent 先前正搬运的所有东西现在都被放到了地上：

```

sitting_at(Obj, Pos, do(drop_everything(Ag), S)) ←
    poss(drop_everything(Ag), S) ∧
    at(Ag, Pos, S) ∧
    carrying(Ag, Obj, S)

```

针对 *carrying* 的框架公理明确了 Agent 在 *drop_everything* 动作后没有搬运任何对象。

```

carrying(Ag, Obj, do(A, S)) ←
    poss(A, S) ∧
    carrying(Ag, Obj, S) ∧
    A ≠ drop_everything(Ag) ∧
    A ≠ putdown(Ag, Obj)

```

603

因此，*drop_everything* 动作能影响的对象数量没有限制。

情景演算通过要求一个使得目标为真的情景被用于规划(planning)。答案抽取被用于

寻找一个使得目标为真的情景。这个情景可解释为 Agent 要做的一系列动作。

【例 14-10】 假设机器人的目标是得到钥匙 $k1$ 。下面的查询是要请求一个使该目标为真的情景：

$?carrying(rob, k1, S)$

这个查询有如下的答案：

```
S=do(pickup(rob, k1),
      do(move(rob, o103, mail),
          do(move(rob, o109, o103),
              init)))
```

这个结果可被解释为 rob 拿到钥匙的一种方式：它从 $o109$ 移动到 $o103$ ，然后移动到 $mail$ ，在这里它拿起了钥匙。

把包裹(初始位置在休息室 lng)运送到 $o111$ 的目标可以通过如下查询来询问：

$?at(parcel, o111, S)$

这个查询有如下的答案：

```
S=do(move(rob, o109, o111),
      do(move(rob, lng, o109),
          do(pickup(rob, parcel),
              do(move(rob, o109, lng), init))))
```

因此， rob 应该去休息室，拿起包裹，回到 $o109$ ，然后再去 $o111$ 。 ◀

在情景演算定义中使用自顶向下的证明过程是非常低效的，因为框架公理几乎总是适用的。一个完整的证明过程，如迭代加深，会搜索动作的所有置换，包括那些与目标无关的置换。答案抽取的使用并不否定使用高效的规划器的必要性，正如第 8 章中的内容那样。

14.1.2 事件演算

第二种表示方法——**事件演算**(event calculus)，建模了关系的真值如何依据发生在某些时间的事件而发生变化。时间可被建模为连续的，也可被建模为离散的。

事件被建模为发生在特定的时间。事件 E 发生在时间 T 可记为 $event(E, T)$ 。

事件使一些关系为真，使另一些关系不再为真：

- $initiates(E, R, T)$ 为真，如果事件 E 在时间 T 使原始关系 R 为真；
- $terminates(E, R, T)$ 为真，如果事件 E 在时间 T 使原始关系 R 不再为真。

时间 T 是 $initiates$ 和 $terminates$ 的一个参数，因为一个事件的影响还依赖于当时其他一些因素是否为真。例如，试图打开一扇门的结果依赖于机器人的位置及它是否拿着合适的钥匙。

在任何时刻，关系非真即假。在事件演算中，关系被具体化了，其中用 $holds(R, T)$ 表示关系 R 在时间 T 为真。这与情景演算中 T 作为 R 最后一个参数是类似的。元谓词 $holds$ 的使用适用于对所有关系都为真的一般规则。

导出关系可以通过同时使用原始关系和其他导出关系来定义。

如果一个发生在 T 时间之前的事件使关系 R 为真，并且没有干扰事件使 R 不再为真，则原始关系 R 在时间 T 为真。这可以说明如下：

```

holds(R, T) ←
    event(E, T0) ∧
    T0 < T ∧
    initiates(E, R, T0) ∧
    ~clipped(R, T0, T)
clipped(R, T0, T) ←
    event(E1, T1) ∧
    terminates(E1, R, T1) ∧
    T0 < T1 ∧
    T1 < T

```

原子 *clipped*(*R*, *T*₀, *T*) 意味着在时间 *T*₀ 和 *T* 之间发生了一个事件使得 *R* 不再为真；*T*₀ < *T*₁ 为真，如果时间 *T*₀ 在时间 *T*₁ 之前。这里，~ 表示否定即失败，因此这些子句意味着它们的完备性。

对于动作，可用它们的激发的属性和终止的属性来表示。在情景演算中，动作的前件用 *poss* 关系来限定。

【例 14-11】 只要动作 *pickup* 的前件为真，则它从激发 *carrying* 关系开始，并终止于 *sitting_at* 关系：

```

initiates(pickup(Ag, Obj), carrying(Ag, Obj), T) ←
    poss(pickup(Ag, Obj), T)
terminates(pickup(Ag, Obj), sitting_at(Obj, Pos), T) ←
    poss(pickup(Ag, Obj), T)
poss(pickup(Ag, Obj), T) ←
    autonomous(Ag) ∧
    Ag ≠ Obj ∧
    holds(at(Ag, Pos), T) ∧
    holds(sitting_at(Obj, Pos), T)

```

这意味着当动作 *pickup* 的前件不成立时，执行该动作将不会发生任何事情。也可以用一些子句来说明在不同的环境下执行该动作会发生什么，例如试图拿起一个被其他物件持有的对象时所发生的情况。

如果出现了特定的动作，且完全知识假设成立（所有涉及的事件均已明确），则带有失败即否定的自顶向下的证明过程可以用来证明哪些为真。对于规划问题，Agent 能够使用溯因推理。

14.2 个体与关系的学习

在个体和关系的学习方面，其中关系的结构对于预测是很重要的。第 7 章讲述的学习方法均假设特征值都是有意义的；通过使用特征值，可以预测一种情况的一些有用信息。然而，当样本是个体和关系时，属性的值可能是个体的无意义的名字。只有通过考虑名称所指的个体的属性及该个体与其他个体的关系，Agent 才能做出有意义的预测。学习到的知识可以应用到未出现在训练集中的个体上。

由于这项工作的很大部分已在逻辑编程的环境下完成，因此通常称之为归纳逻辑编程 (inductive logic programming)。

【例 14-12】 假设一个交易 Agent 有如下关于人们喜欢的度假胜地的数据集，数据条

目用个体-属性-值(individual-property-value)三元组的形式表示：

个体	属性	值
joe	likes	resort_14
joe	dislikes	resort_35
...
resort_14	type	resort
resort_14	near	beach_18
beach_18	type	beach
beach_18	covered_in	ws
ws	type	sand
ws	color	white
...

该 Agent 想知道 Joe 喜欢什么。此时，重要的不是 *likes* 属性的值——这个值只是一个没有意义的名称——而是个体通过该名称所指的属性。除能了解到 Joe 喜欢 *resort_14* 不喜欢 *resort_35* 之外，基于特征的表示不能在这个数据集上做任何事情。

可以猜测，Agent 应该学习的是 Joe 喜欢靠近海边沙滩的旅游胜地。这个猜测可以表示为一个逻辑程序：

```
prop(joe, likes, R) ←
    prop(R, type, resort) ∧
    prop(R, near, B) ∧
    prop(B, type, beach) ∧
    prop(B, covered_in, S) ∧
    prop(S, type, sand)
```

逻辑程序提供了表示关于个体和关系的确切猜测的能力。这个规则可以应用于 Joe 没有去过的旅游胜地和海滨。 ◀

归纳逻辑编程学习器的输入内容如下：

- 原子集合 A ，其定义了 Agent 的学习内容；
- E^+ 是 A 元素的基实例的集合，称为正类样本(positive example)，值为真；
- E^- 是 A 元素的基实例的集合，称为负类样本(negative example)，值为假；
- B 是背景知识(background knowledge)，是定义了能用于逻辑程序学习结果的关系的子句集合；
- H 是一个可能假设空间。 H 经常隐式地表示为一个算子的集合，该算子可生成可能的假设。我们假定每一个假设都是一个逻辑程序。

607

【例 14-13】 在例 14-12 中，假设 Agent 想学习 Joe 喜欢什么。此时的输入是：

- $A = \{prop(joe, likes, R)\}$ 。
- $E^+ = \{prop(joe, likes, resort_14), \dots\}$ 。对于接下来的例子，我们假设有很多 Joe 喜欢的类似项目。
- $E^- = \{prop(joe, likes, resort_35), \dots\}$ 。请注意，这里用正类格式来写；它们的观察值为假。
- $B = \{prop(resort_14, type, resort), prop(resort_14, near, beach_18), \dots\}$ 。这个集合包含了所有不在 A 中的关于环境的事实。Agent 不学习这些内容。

- H 是一个定义了 $prop(joe, likes, R)$ 的逻辑程序的集合。子句的头部与 $prop(joe, likes, R)$ 一致。集合 H 太大而不能枚举出来。

以上除了 H 之外，都在问题的说明中明确给出。

现在的目标是找到一个最简单的假设 $h \in H$ 满足：

$$B \wedge h \models E^+ \text{ 且}$$

$$B \wedge h \not\models E^-$$

也就是说，假设蕴含着正证据且不蕴含负证据。假设必须与负证据为假相容(consistent)。

因此，我们希望找到变体空间中的一个元素。这类似于溯因的定义，在那里溯因知识库对应于背景知识。归纳逻辑编程的假设空间是逻辑程序的集合。第二个条件对应于相容性。

本节的其余部分假设我们想要学习的是一个单目标的 n 元关系的定义，用谓词符号 t 来表示，即 $A = \{t(X_1, \dots, X_n)\}$ 。关系的假设空间由可以使用逻辑程序给出的所有定义形式组成。

归纳逻辑编程中主要使用下面两个策略：

- 第一个策略是从最简单的假设开始，然后逐渐变得复杂以适应数据。因为逻辑程序仅规定了正类事实，所以最简单的假设是最一般的假设，仅简单表明 $t(X_1, \dots, X_n)$ 总为真。这个假设蕴含了正类样本，但它也蕴含了负类样本(如果有的话)。该策略包含了从一般到具体的搜索。它试图在假设空间中通过从最简单的假设到更加复杂的假设的搜索找到一个适应数据的最简单的假设；搜索的假设总是蕴含正类样本，且该搜索直到找到一个不蕴含负类样本的假设时才停止。
- 第二个策略是从一个适合数据的假设开始，然后在保证仍然适合数据的情况下不断简化。适合数据的假设是正类样本的集合。这一策略包含了从具体到一般的搜索：从最具体的假设开始——该假设中仅有正类样本为真，然后推广子句，避免涵盖负类样本。

我们将在从一般到具体的搜索中扩展限定子句。初始假设包含了一个子句：

$$\{t(X_1, \dots, X_n) \leftarrow\}$$

具体化算子(specialization operator)观察子句集合 G 并返回具体化 G 之后的子句集合 S 。具体化表示为 $S \models G$ 。

以下是三个原始的具体化算子：

- 在条件 c 下拆分 G 中的一个子句。集合 G 中的子句 $a \leftarrow b$ 被两个子句代替： $a \leftarrow b \wedge c$ 和 $a \leftarrow b \wedge \neg c$ 。
- 根据出现在 a 或 b 中的变量 X 拆分 G 中的子句 $a \leftarrow b$ 。子句 $a \leftarrow b$ 被以下子句代替：
 $a \leftarrow b \wedge X = t_1$
 \dots
 $a \leftarrow b \wedge X = t_k$
 其中 t_i 是项。
- 删除证明正类样本非必需的子句。

最后一个操作改变了子句集的预言。那些不再被子句集合蕴含的情况为假。

这些原始的具体化算子结合使用，形成了 H 中的算子。集合 H 中的算子设计为原始具体化算子的组合，因此可以使用贪婪(greedy)前瞻的方法来评估搜索的进展。也就是说，对算子进行了定义，从而使得一步即可评估搜索的进展情况。

前两个原始的具体化操作应该明智而谨慎地执行，以确保发现最简单的假设。Agent

只在拆分操作能取得进展的情况下才执行拆分操作。拆分操作只在与子句删除操作结合使用时才有用。例如，向子句添加一个原子相当于根据该原子进行拆分操作，然后删除含有否定该原子的子句。更高层次的具体化算子可能为一些出现在子句中的变量 X 根据原子 $prop(X, type, T)$ 执行拆分操作：根据值 T 进行拆分，然后删除对于蕴含正类样本来说不必要的子句。这个算子在确定有用的类型方面做出改进。

图 14-1 给出了一个用于逻辑程序自顶向下归纳的局部搜索(local search)算法。该算法维护了一个假设，并反复改进它直到找到一个适合数据的假设(或者直到无法找到一个这样的假设)才停止。

609

```

1: procedure TDInductiveLogicProgram( $t, B, E^+, E^-, R$ )
2:   Inputs
3:      $t$ : 一个原子，其定义为学习目标
4:      $B$ : 背景知识，是一个逻辑程序
5:      $E^+$ : 正类样本
6:      $E^-$ : 负类样本
7:      $R$ : 具体化算子集合
8:   Output
9:     将  $E^+$  分类为正、 $E^-$  分类为负的逻辑程序，当找不到上述逻辑程序时输出  $\perp$ 
10:  Local
11:     $H$  是子句的集合
12:     $H \leftarrow \{t(X_1, \dots, X_n) \leftarrow\}$ 
13:    while 存在  $e \in E^-$  满足  $B \cup H \models e$  do
14:      if 存在  $r \in R$  满足  $B \cup r(H) \models E^+$  then
15:        选择  $r \in R$  满足  $B \cup r(H) \models E^+$ 
16:         $H \leftarrow r(H)$ 
17:      else
18:        return  $\perp$ 
19:    return  $H$ 

```

图 14-1 逻辑程序的自上顶向归纳

在每一个时间步，算法都在每一个假设都蕴含了正类样本的约束下选择一个算子。该算法掩盖了两个重要的细节：

- 需要考虑哪些算子。
- 最终选择哪个算子。

这些算子应处在一定的水平，以便可以根据是否向找到一个好的假设方向取得进展来评价它们。与决策树学习中的方式类似，该算法可以执行当前最优的选择。也就是说，它在使得误差最小的算子中选择使得进展最大的那个算子。假设的误差可以是该假设所蕴含的负类样本的数量。

【例 14-14】 考虑例 14-12。在这个例子中，Agent 必须学习 Joe 喜欢什么、不喜欢什么。

第一个假设是 Joe 喜欢所有的事情：

$\{prop(joe, likes, R) \leftarrow\}$

该假设与负类证据不相容。

可以根据包含了 Joe 喜欢的个体的属性来拆分该假设。为取得进展(即在蕴含了所有正类样本的前提下包含了更少的负类样本)的具体化方法是把变量 R 包含在具体化结果中。因此，必须考虑 R 在三元组中作为主语或宾语的那些三元组。

610

- 可以考虑在属性 $type$ 上进行拆分：根据 $type$ 的取值进行拆分，并只保留那些为证明正类样本所需要的内容。最后得到下面的子句(假设正类样本只包括旅游胜地 *resort*)：

$$\{prop(joe, likes, R) \leftarrow prop(R, type, resort)\}$$

如果正类样本包含除 *resort* 之外的其他东西(Joe 也喜欢这些东西), 则也可有其他子句。如果负类样本包含了非旅游胜地的内容, 则这个拆分将有助于减少误差。

- 也可以考虑根据有助于证明正类样本的其他属性进行拆分, 例如如果所有的正类都在某物附近, 根据 *near*(接近)属性进行拆分可得

$$\{prop(joe, likes, R) \leftarrow prop(R, near, B)\}$$

如果有一些负类样本不在任何事物附近, 则这个具体化将有助于减少误差。

然后可以在这些拆分操作中选择使进展最大者。在下一步, 可以增加另一子句、*R* 的其他属性或 *B* 的其他属性。

14.3 概率关系模型

第 6 章中的信念网络概率模型是依据特征定义的。许多领域依据个体和关系有了最好的模型。Agent 常常必须在它们知道领域中有哪些个体之前——从而也在它们知道存在哪些随机变量之前——建立概率模型。在学习到上述概率之后, 这些概率常常不依赖于个体。尽管学习某一个体是可能的, 但 Agent 必须同时学习一般的知识, 从而, 在 Agent 发现新的个体时能够运用这些知识。

【例 14-15】 考虑一个智能教学系统诊断学生的算术错误问题。通过观察学生在几个练习题上的表现, 教师应努力确定学生是否理解了学习内容, 如果没有理解, 则应找出学生哪里做错了, 从而能够采用适当的补救措施。

考虑诊断两位数加法算式的情况:

$$\begin{array}{r} x_1 \quad x_0 \\ + \quad y_1 \quad y_0 \\ \hline z_2 \quad z_1 \quad z_0 \end{array}$$

给了学生各个 x 的值和各个 y 的值, 要求学生给出各个 z 的值。

学生的答案依赖于他们是否知道基本的加法 (*Knows_Add*), 以及他们是否知道如何进位 (*Knows_Carry*)。这个例子的信念网络如图 14-2 所示。第 i 位的进位, 用变量 C_i 表示, 依赖于 x 的值、 y 的值、前一位数字的进位, 及学生是否知道如何进位。对于第 i 位的 z 值, 表示为变量 Z_i , 依赖于 x 的值、 y 的值、第 i 位上的进位, 及学生是否知道基本的加法。

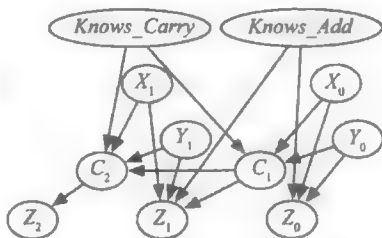


图 14-2 两位数加法的信念网络

通过观察问题中 x 的值和 y 的值, 以及学生给出的 z 的值, 可以推断出学生知道基本的加法及如何进位的后验概率。信念网络模型的特征之一是允许学生做出随机的错误; 即使他们知道算法, 也仍可能得到错误的答案。

使用这种表示方式的问题在于不够灵活。一个灵活的表示方式会适应多位数字的加法、多类的问题、多个学生及多次时间。多位数字要求网络根据数字位数进行复制。多次时间要求对学生的知识及答案如何随着时间而改变进行建模, 即使上述问题并不随时间而改变。

如果条件概率被储存为表格, 则这些表格的规模将是巨大的。例如, 如果每一个变量 X_i 、 Y_i 和 Z_i 的域大小均为 11(数字 0~9 或空白 *blank*), C_i 和 *Knows_Add* 变量为二进

制, 则下式的表格形式

$$P(Z_i | X_i, Y_i, C_i, \text{Knows_Add})$$

的大小将会超过 4000。除使用表格表示外, 还有许多表示条件概率的结构。下面要介绍的结构不仅可以适用于关系概率模型, 而且可以适用于条件概率的压缩描述。

概率关系模型 (Probabilistic Relational Model, PRM) 或 **关系概率模型** (relational probability model) 是一种在关系上明确了概率值, 且概率独立于实际个体的一种模型。不同的个体共享概率参数。

612

参数化随机变量 (parametrized random variable) 或者是一个逻辑原子, 或者是一个项。也就是说, 它的形式是 $p(t_1, \dots, t_n)$, 其中 t_i 为逻辑变量或者为常量。参数化随机变量之所以被称为参数化的, 是因为出现在其中的逻辑变量。将参数化随机变量中的逻辑变量替换为常量就可得到参数化随机变量的基实例。参数化随机变量的基实例对应于随机变量。如果 p 是一个谓词符号, 则该随机变量是一个布尔变量。如果 p 是一个函数符号, 则该随机变量的值域为 p 的取值范围。

我们这里使用的符号体例为: 逻辑变量用大写字母表示; 随机变量对应于基原子和基项, 因此在本节使用小写字母表示。

【例 14-16】 对于前面例子的多位数字算术问题的 PRM, 都有一个独立的变量 x 对应于其中的每一位数字 D 和每一个问题 P , 该参数化随机变量表示为 $x(D, P)$ 。例如, $x(1, \text{probl7})$ 是一个随机变量, 它表示了问题 17 的第一位 x 的值。类似地, 有一个参数化随机变量 $y(D, P)$ 表示了每一个问题 P 和每一位数字 D 的随机变量。

每一个学生 S 的每一时间 T 对应有一个变量, 该变量表示了 S 在时间 T 是否知道如何做加法。参数化随机变量 $\text{knows_add}(S, T)$ 表示学生 S 在时间 T 是否知道加法运算。如果 fred 在 3 月 23 日知道如何做加法, 则随机变量 $\text{knows_add}(\text{fred}, \text{mar23})$ 为真。类似地, 还有一个参数化随机变量 $\text{knows_carry}(S, T)$ 。

对于每一位数字、每一个问题、每一个学生、每一个时间, 都有不同的 z 值和不同的进位。这些值表示为参数化随机变量 $z(D, P, S, T)$ 和 $\text{carry}(D, P, S, T)$ 。因此, 随机变量 $z(1, \text{probl7}, \text{fred}, \text{mar23})$ 表示了 fred 在 3 月 23 日为问题 17 计算出的第一位数字的值。函数 z 的值域为 $\{0, \dots, 9, \text{blank}\}$, 因此上述随机变量的取值范围均为此集合。

一个平板模型 (plate model) 的组成如下:

- 一个有向图, 其中的节点表示参数化随机变量。
- 每一个逻辑变量有一个种群。
- 每一节点的条件概率由其父节点给出。

平板模型意味着对应信念网络的具体化, 即信念网络中的节点都是参数化随机变量的基实例 (每一个逻辑变量均由对应种群中的一个个体所替代)。信念网络中的条件概率对应于平板模型的实例。

【例 14-17】 图 7-17 给出了一个层次贝叶斯模型, 其中 (图 7-17a) 是一个平板模型, 图 7-17b 是该模型的一个实例。图中左边的平板表示有 4 个参数化随机变量: $s(X, H)$ 、 $\phi(H)$ 、 α_1 和 α_2 。 X 的种群为全体患者的集合。 H 的种群为全体医院的集合。该网络的实例列在图的右边。每一个医院 h 对应于一个单独的随机变量 $\phi(h)$; 每一个病人 x 和医院 h 对应于单独的随机变量 $s(x, h)$ 。对每一个 $s(x, h)$ 给定了对应的 $\phi(h)$ 的条件概率; 对每一个 $\phi(h)$ 给定 α_1 和 α_2 的条件概率; 同时给出了 α_1 和 α_2 的先验概率。实例中的每一个

613

$s(x, h)$ 变量都有相同的由父节点规定的条件概率。对于不同的病人 x_1 和 x_2 ，给定了 $\phi(h)$ 的变量 $s(x_1, h)$ 和 $s(x_2, h)$ 是相互独立的。对于不同的医院 h_1 和 h_2 ，给定了 α_1 和 α_2 时的变量 $s(x_1, h_1)$ 和变量 $s(x_2, h_2)$ 是独立的。

【例 14-18】 例 14-16 的带参数化随机变量的平板模型如图 14-3 所示。其中的矩形对应于平板。对于标记为 D, P 的平板，每一个变量都有对应于每一位数字 D 和每个人 P 的实例。从某个角度来看，这些实例来自于历史记录，就像堆积的一摞盘子一样。类似地，对于标记为 S, T 的平板，对于每一个学生 S 和每一个时间 T ，都有一个变量的副本。对于两平板重叠区域的

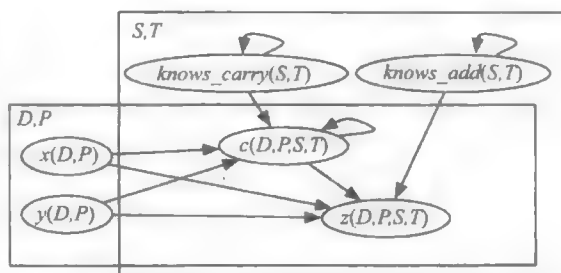


图 14-3 多位数字加法的带平板的信念网络

变量，对于每一个数字 D 、每一个人 P 、每一个学生 S 和每一个时间 T ，都对应有一个随机变量。请注意，这里的表示是冗余的；平板提供的信息与明确的参数提供的信息是相同的； c 的参数一定是 D, P, S 和 T ，因为它们含有 c 的平板。

平板表示方式表示了与信念网络相同的独立性；给定了父节点，每个节点都独立于其非子孙节点。在这种情况下，网络图就描述了平板的基础。因此，对于特定的 $d \in \text{dom}(D)$ 、 $p \in \text{dom}(P)$ 、 $s \in \text{dom}(S)$ 及 $t \in \text{dom}(T)$ ， $z(d, p, s, t)$ 是一个父节点为 $x(d, p)$ 、 $y(d, p)$ 、 $c(d, p, s, t)$ 和 $\text{knows_add}(s, t)$ 的随机变量。在平板模型中的参数化随机变量 $c(D, P, S, T)$ 上有一个循环，因为对于相同的问题、学生和时间，一位数的进位也依赖于前一位数的进位。类似地，在某一时间学生是否知道如何进位依赖于他们之前是否知道如何进位。然而，条件概率表使得基础网络是无环的。

给定了父节点，每个参数化随机变量都有一个条件概率。这个条件概率在它的基础实例中是共享的。

不幸的是，当相同关系下的不同实例中产生相互依赖关系时，仅靠平板表示是不够的。在之前的例子中， $c(D, P, S, T)$ 部分依赖于 $c(D-1, P, S, T)$ ，即部分依赖于前一位数字的进位（对于第 1 位数字来说还有其他的情况）。另一个更复杂的例子是确定两个作者是否是合作者的概率，确定的依据是他们是否共同写过一篇文章。为了表示这类情况，能够明确逻辑变量间的交互是有用的，正如在逻辑程序中做的那样。

一种综合了信念网络、平板和逻辑程序思想的表示是**独立选择逻辑**（Independent Choice Logic, ICL）。ICL 由一组独立选择、为各个选择指定结果的逻辑程序及在选择上的概率分布组成。更具体的，ICL 的定义如下：

抉择（alternative）是原子的集合，这些原子共享相同的逻辑变量。**选择空间**（choice space）是抉择的集合，其中抉择中的原子互不合一。ICL 理论包含：

- 一个选择空间 C 。设 C' 为抉择的基实例集合。因此， C' 是基原子集合的集合。
- 一个无环逻辑程序（它可以包含否定即失败推理），其中，子句的头部与选择空间中的抉择的元素不合一。
- 在每一个抉择上的概率分布。

逻辑程序和选择空间中的原子可以包括常数、变量和函数符号。

选择器函数（selector function）从 C' 的每一个抉择中选择一个元素。对于每一个选择器函数存在一个**可能世界**（possible world）。逻辑程序明确了在每一个可能世界中哪些为

真。在一个可能世界中，原子 g 为真，如果它是由选择器函数选出并加入逻辑程序的原子之一。命题 g 的概率通过在可能空间集合上的一个度量给出，其中各抉择的不同基础实例中的原子是概率独立的。一个抉择的实例共享相同的概率，且不同实例的概率相乘。

逻辑程序的无环性意味着对于涉及证明任一个 g 的原子，仅存在有限个抉择。因此，尽管对于任意的 g 可能有无限多的可能世界，概率度量仍可在可能世界集合上定义，而这些可能世界可通过有限的抉择来描述。这些世界的描述可通过下面的方式找出来：使原子在抉择中可假定，借助不同的原子在相同的抉择中产生不一致，并使用溯因推理来发现在使 g 为真的世界集合的描述。

615

ICL 理论可被看做一个因果模型，在该模型中，因果机制使用逻辑程序表示，而背景变量——对应于各个抉择——则在各抉择上分配独立的概率数。可能有人认为，这种只有无条件独立的原子和确定的逻辑程序的逻辑在表达需要的知识类别上显得能力太弱。然而，即使不用逻辑变量，独立选择逻辑也能表示出信念网络所能表示的所有概念。

【例 14-19】 考虑使用独立选择逻辑表示例 6-10 中的信念网络。这个技术适用于任何信念网络。

火情和干扰没有父节点，因此，它们可以被直接表示成抉择：

$\{fire, nofire\}$

$\{tampering, notampering\}$

在第一个抉择上的概率分布为 $P(fire)=0.01$, $P(nofire)=0.99$ 。类似的, $P(tampering)=0.02$, $P(notampering)=0.89$ 。

Smoke 和 Fire 间的依赖关系可使用两个抉择来表示：

$\{smokeWhenFire, nosmokeWhenFire\}$

$\{smokeWhenNoFire, nosmokeWhenNoFire\}$

其中, $P(smokeWhenFire)=0.9$, $P(smokeWhenNoFire)=0.01$ 。可用两个规则来说明什么时候有烟：

$smoke \leftarrow fire \wedge smokeWhenFire$

$smoke \leftarrow \sim fire \wedge smokeWhenNoFire$

为表示 Alarm 如何依赖于 Fire 和 Tampering，可用如下 4 个抉择：

$\{alarmWhenTamperingFire, noalarmWhenTamperingFire\}$

$\{alarmWhenNoTamperingFire, noalarmWhenNoTamperingFire\}$

$\{alarmWhenTamperingNoFire, noalarmWhenTamperingNoFire\}$

$\{alarmWhenNoTamperingNoFire, noalarmWhenNoTamperingNoFire\}$

其中, $P(alarmWhenTamperingFire)=0.5$, $P(alarmWhenNoTamperingFire)=0.99$ ，类似地，其他原子也使用例 6-10 中的概率。同样还有明确 alarm 何时为真的规则 (alarm 何时为真依赖于干扰和火情)：

$alarm \leftarrow tampering \wedge fire \wedge alarmWhenTamperingFire$

$alarm \leftarrow \sim tampering \wedge fire \wedge alarmWhenNoTamperingFire$

$alarm \leftarrow tampering \wedge \sim fire \wedge alarmWhenTamperingNoFire$

$alarm \leftarrow \sim tampering \wedge \sim fire \wedge alarmWhenNoTamperingNoFire$

其他随机变量的表示方法类似，可使用与父节点的取值数目相同个数的抉择来表示。 ◀

616

条件概率的 ICL 表示可以被看做决策树规则的形式，其中概率在叶子节点上确定。每一个分支都对应一个规则和一个抉择。当涉及非二进制的变量时，非二路的抉择就很

有用。

在表示标准的信念网络时，独立选择逻辑可能不是非常直观，但它可以使复杂的关系模型变得简单，如下面的例子所示。

【例 14-20】 考虑例 14-16 中多位数加法的参数化版本。平板对应于逻辑变量。

$z(D, P, S, T)$ 的取值有三种情况。第一种情况是学生在这个时间知道加法运算，并且回答没有错误。此时，他们得到了正确的答案：

```

 $z(D, P, S, T, V) \leftarrow$ 
   $x(D, P, V_x) \wedge$ 
   $y(D, P, V_y) \wedge$ 
   $carry(D, P, S, T, V_c) \wedge$ 
   $knowsAddition(S, T) \wedge$ 
   $\sim mistake(D, P, S, T) \wedge$ 
   $V \text{ is } (V_x + V_y + V_c) \text{ div } 10$ 

```

在这里，原子 $z(D, P, S, T, V)$ 用来表示参数化随机变量 $z(D, P, S, T)$ 取值为 V 。这是一个标准的逻辑规则，也与限定子句的含义相同。

这里有一种针对学生是否在上述情况下碰巧犯了错误的抉择：

```

 $\forall P \forall S \forall T \{ noMistake(D, P, S, T), mistake(D, P, S, T) \}$ 

```

其中， $mistake(D, P, S, T)$ 的发生概率为 0.05，即假设学生在知道如何做算术的情况下仍然有 5% 的概率做错。

第二种情况是学生在这个时间知道加法运算但却犯了错误。此时，我们假设学生在每位数字上犯错的可能性是相同的：

```

 $z(D, P, S, T, V) \leftarrow$ 
   $knowsAddition(S, T) \wedge$ 
   $mistake(D, P, S, T) \wedge$ 
   $selectDig(D, P, S, T, V)$ 

```

这里有一个抉择指明学生选取了哪个数字：

```

 $\forall P \forall S \forall T \{ selectDig(D, P, S, T, V) \mid V \in \{0..9\} \}$ 

```

对于每一个 v ，假设 $selectDig(D, P, S, T, v)$ 的概率为 0.1。

最后一种情况为学生不知道如何做加法。此时，学生随机地选择一个数字：

```

 $z(D, P, S, T) = V \leftarrow$ 
   $\sim knowsAddition(S, T) \wedge$ 
   $selectDig(D, P, S, T) = V$ 

```

这三种规则覆盖了 z 的所有规则；它比用表格表示所需要的、大于 4000 条数据的表格要简单得多，并且它适用于任意位数字、任意问题、任意学生和任意的时间。不同的数字和问题为 $x(D, P)$ 赋予不同的值，且对于不同的学生和时间有不同的值来表示他们是否知道加法运算。

对于进位的规则是类似的，其主要的不同在于规则体中的进位依赖于前一数字。

在任意时间，学生是否知道加法运算依赖于他们在前一时间是否知道加法运算。根据推测，学生的知识还依赖于发生了哪些动作（学生和老师做过什么）。因为 ICL 允许标准逻辑程序（带有“噪声”），所以在本章开始介绍的表示方法均可用来表示模型的变化。◀

前面章节用过的 AILog 也实现了 ICL。

14.4 本章小结

- 当 Agent 要求在它知道将会遇到什么个体之前给出或学习模型时，就会用到关系表示。

- 许多之前章节出现过的表示方式可以转换成关系的形式。
- 在情景演算中, 根据 Agent 的动作, 使用 *init* 常量和 *do* 函数来表示时间。
- 事件演算允许有连续和离散的时间, 并公理化事件发生后的后续事情。
- 归纳逻辑编程可以用来学习关系模型, 即使特征的值是无意义的名字也可以。
- 独立选择逻辑允许在个体被认知之前详细说明概率模型。

14.5 参考文献及进一步阅读

情景演算由 McCarthy 和 Hayes[1969]提出。这里提出的框架公理的形式可以追溯到 Kowalski[1979]、Schubert[1990]及 Reiter[1991]。Reiter[2001]对情景演算做了一个全面的总结; Brachman 和 Levesque[2004]也做了这个工作。关于如何解决框架问题, 即简明扼要地阐述在执行一个动作的过程中那些不变的内容, 研究者还提出了许多其他的建议。事件演算由 Kowalski 和 Sergot[1986]提出。Shanahan[1997]为表示改变所涉及的问题和特定的框架问题做了一个极好的介绍。

618

归纳逻辑编程的概述可以参见 Muggleton 和 DeRaedt[1994]、Muggleton[1995]及 Quinlan 和 Cameron-Jones[1995]。

独立选择逻辑由 Poole[1993, 1997]提出。De Raedt、Frasconi、Kersting 和 Muggleton[2008]及 Getoor 和 Taskar[2007]的论文概述了概率关系模型及如何学习这个模型。

关系、归属及存在的不确定性

在 14.3 节介绍的模型关注的是**关系的不确定性**, 即有关一些个体间的关系是否为真的不确定性。例如, 我们可以设计一个 *likes*(X, Y) 的概率模型, 其中 $X \neq Y$, 它为不同的人相互喜欢建立了模型。这可能依赖于 X 和 Y 的外部特征。我们也可以为 *friends*(X, X) 设计一个概率模型, 它为某个人是否喜欢自己建立了模型。这可能依赖于该人的内部特征。这样的模型对于辅导系统确定两个人是否应该在一起工作, 或者确定导致学生犯错误的原因是材料的难度还是学生的个性, 是很有用的。

给定个体 *sam* 和 *chris*, 只有在我们知道 Sam 和 Chris 是不同的人时我们才能使用概率模型 *likes*(*sam*, *chris*)。**归属不确定性**(identity uncertainty)问题关注的是两个项是否指示的是同一个体的不确定性。这对于医疗系统是一个问题, 在该系统中, 确定正在与该系统交互的人是否就是昨天已经访问过的那个人是很重要的。如果病人本就不愿沟通或者想欺骗该系统(比如想得到毒品), 则识别这个问题就尤为困难。在医学界, 该问题就是**记录关联**(record linkage)问题, 其目标是确定哪些医疗记录是同一个人的。

在上面的例子中, 已知个体是存在的; 给定名字 Chris 和 Sam 就是说假定他们存在。给定一个描述, 确定是否存在一个个体符合该描述的问题就是**存在不确定性**(existence uncertainty)问题。存在不确定性之所以成为一个问题是因为可能不存在一个符合该描述的个体, 或者有多个个体符合该描述。我们不能给一个不存在的个体赋予属性, 因为不存在的个体根本就没有属性。如果我们想给一个存在的个体指定一个名字, 则我们需要关心的是我们指的是哪个个体(如果存在多个个体)。存在不确定性的一个特例是**数量不确定性**(number uncertainty), 即存在的个体数量不确定。例如, 一个采购 Agent 可能无法确定对参加旅游团感兴趣的人数, 而旅游团能否成行取决于对此感兴趣的人的数量。

如果涉及复杂的角色, 且问题是确定是否有个体适合这些角色, 则关于存在不确定性的推理会非常棘手。考虑一个采购 Agent——它必须为 Sam 和她的儿子 Chris 找到一个公寓。Sam 是否喜欢一个公寓的概率部分依赖于她的房间的大小和 Chris 的房间的颜色。然而, 个人的公寓并不会标记 Sam 的房间和 Chris 的房间, 且可能并不存在他们所希望的房间。给定一个 Sam 希望的公寓情况模型, 如何在观察上平衡条件仍然是不明确的。

14.6 习题

14.1 给情景演算的例子(可从本书网站上得到)添加为一个物体涂颜料的能力。特别地, 添加谓词

color(Obj, Col, Sit)

如果物体 *Obj* 在情景 *Sit* 时的颜色为 *Col* 则该谓词为真。

包裹的颜色一开始是蓝色。因此，我们有一个公理：

color(parcel, blue, init)

有一个动作 *paint(Obj, Col)* 可把物体 *Obj* 涂成 *Col* 颜色。对于这个问题，假设物体仅可被涂为红色，且仅在物体和机器人均位于位置 *o109* 时才能够涂色。颜色在机器人上有积累效果（没办法把颜色从一个物体上去除，即如果你把包裹涂成红色，则包裹既是红色又是蓝色——当然，这在现实中是不存在的，但这样做可使问题得到简化）。

使用情景演算公理化谓词 *color* 和动作 *paint*。

你可以使用不超过三个子句做到这一点（包括前面提到的定义了初始情景颜色的子句），其中任一个子句的主体中都不超过两个原子符号。你不需要相等、不等或否定即失败推理。请在 AILog 中测试。

你得到的输出应该大致如下：

```
ailog: bound 12.
ailog: ask color(parcel, red, S).
Answer: color(parcel, red, do(paint(parcel, red),
                             do(move(rob, storage, o109),
                                do(pickup(rob, parcel),
                                   do(move(rob, o109, storage),
                                      init))))))
```

14.2 在这个问题中，你将添加一个比前一习题中更加复杂的涂颜色的动作。

假设对象 *paint_can(Color)* 表示一罐颜色为 *Color* 的颜料。

添加一个动作 *paint(Obj, Color)*，它可将物体的颜色涂成 *Color* 色（与前一个习题不同，此时物体仅有一个颜色）。涂颜色的动作只有在物体位于 *o109*、自主 Agent 也位于 *o109* 且携带着一罐合适颜色的颜料时才能进行。

14.3 AILog 执行深度有限搜索。你可能已注意到求解前面的问题时比较慢，因此我们要求一个深度限制（该限制的深度接近于实际的深度）以使问题的求解时间限制在合理的范围之内。

在此，估算下面的查询需要多长时间的迭代加深搜索才能找到一个解：

ask sitting_at(parcel, lab2, S)

（不要刻意去尝试——它的运行时间非常长。）

- 估计能找到一个规划的最小的限制。（提示：解决这个问题需要多少步？这些步数与需要的深度限制有何关系？）证明你的估计。
- 估算搜索树的分支因子。为做到这一点，你应该考察完成第 $k+1$ 层搜索的时间，并对比完成第 k 层搜索的时间。你既需要通过实验验证你的答案（通过运行程序），也需要通过理论分析证明它（通过考察分支因子的大小）。你不必运行需要大量时间的例子来求解这个问题。
- 根据你给出的前两问的答案，及你在一些小的限定下运行程序时的发现，估计搜索完搜索树需要的时间，其中搜索深度比发现一个解所必需的深度小 1。证明你的答案。

14.4 在这个问题中，你将考察为机器人传送问题使用事件演算。

- 使用事件演算表示动作 *move*。
- 使用事件演算表示例 14-10 中的每一个动作序列。
- 说明事件演算可从 (b) 部分给出的动作序列中导出合适的目标。

14.5 假设在事件演算中有两个动作 *Open* 和 *Close*，及在时间 0 时的值为假的初始关系 *opened*。动作 *Open* 使 *opened* 关系变成真，动作 *Close* 使 *opened* 关系变成假。假设动作 *Open* 发生在时间 5，动作 *Close* 发生在时间 10。

- 使用事件演算表示该问题。
- 关系 *opened* 在时间 3 为真吗？给出推导过程。
- 关系 *opened* 在时间 7 为真吗？给出推导过程。

(d) 关系 *opened* 在时间 13 为真吗? 给出推导过程。

(e) 关系 *opened* 在时间 5 为真吗? 请解释。

(f) 关系 *opened* 在时间 10 为真吗? 请解释。

(g) 给出 *holds* 的可选公理化表示, *holds* 在时间 5 和时间 10 有不同的行为。

(h) 说明其中的一个公理化比另一个好。

621

- 14.6 举出一些具体的可用于自顶向下归纳逻辑编程的具体化算子。定义这些算子, 从而可评价算子的进展。请解释在什么情况下这些算子会取得进展。
- 14.7 对于例 14-20 中加法的表示, 之前假设 z 的观察值均为数字。请改变表示方式, 从而使得观察值可以为数字、空白或其他。请给出合适的概率。
- 14.8 使用 ICL 表示前面章节的电气领域问题, 使得表示可以在 AILog 中运行。这个表示应包括例 6-11 中的概率依赖和例 12-11 中的关系。

622

第五部分

Artificial Intelligence: Foundations of Computational Agents

宏观图景

回顾与展望

计算是我们现代洞穴的火种。到 2056 年，计算革命带来的社会转化将被认为和工业革命带来的转化一样显著。计算的演化和广泛传播所带来的后果将对社会经济学、科学和文化产生重大的影响。

——Eric Horvitz[2006]

本章对前面所讨论的内容作一个宏观的评论。通过将众多的表示方案置于我们先前介绍的设计空间中，各种表示之间的关系就变得明朗了。这样做使得我们能够理解人工智能研究的现状及其演化。我们也将讨论由于智能计算 Agent 的发展和应用所带来的一些社会与道德后果。正如 Horvitz 在引言中所指出的那样，计算正在改变世界；我们必须意识到它的正面与负面影响。

15.1 复杂性维度回顾

人工智能研究的进展如何？目前的前沿问题是什么？为了系统地讨论这些宏观问题，我们利用 1.5 节所介绍的人工智能系统的设计空间来说明。在 1.5 节中，我们介绍了横跨部分设计空间的 9 个维度。本节说明如何将本书中介绍的一些表示方法置于设计空间之中。

图 15-1 回顾了复杂性的维度，它是在图 1-6 的基础上扩展而来的，增加了一个键属性（取值为多个 * 号，在后续的图中会用到）。

维	键	值
组件方式	*	无模块划分
	**	模块化的
	***	分层的
表示方案	*	状态
	**	特征
	***	关系
规划水平	*	无规划
	**	有限期
	***	不定期
	****	无限期
感知的不确定性	*	完全可观察
	**	部分可观察
结果的不确定性	*	确定性的
	**	随机的
偏好	*	得分
	**	复杂的偏好
学习	*	知识是给定的
	**	知识是学得的
Agent 的个数	*	单个 Agent
	**	多个 Agent
计算局限性	*	完全理性
	**	有限理性

图 15-1 复杂性维度

623
/
625

根据各个维度的值, 图 15-2 对本书中介绍的一些表示方法进行了分类。

维	组件方式	表示方案	规划水平	感知的不确定性	结果的不确定性	偏好	学习	Agent 的个数	计算局限性
状态空间搜索	*	*	***	*	*	*	*	*	*
回归规划	*	**	***	*	*	*	*	*	*
约束满足问题规划	*	**	***	*	*	*	*	*	*
决策网络	*	**	***	*	*	*	*	*	*
马尔可夫决策过程	*	*	****	*	*	*	*	*	*
部分可观察马尔可夫决策过程	*	*	****	*	*	*	*	*	*
动态决策网络	*	**	****	*	*	*	*	*	*
多 Agent 决策网络	*	**	***	*	*	*	*	*	*
策略改进	*	*	*	*	*	*	*	*	*
增强学习	*	**	****	*	*	*	*	*	*
情景演算	*	***	***	*	*	*	*	*	*
独立选择逻辑	*	***	***	*	*	*	*	*	*

图 15-2 依据复杂性的维度列出的一些表示方法

第 3 章介绍的状态空间搜索除了允许不定期的规划外, 其他维度均取最简单的值。利用基于特征的表示法或者 STRIP 表示法, 回归规划扩展了状态空间搜索, 以便进行基于特征的推理。约束满足问题(CSP)规划的效率更高, 但代价是只能进行有限期的规划。

基于实用性原则, 决策网络具有下列特性: 基于特征的表示、随机的结果、部分可观察和复杂的偏好。然而, 和约束满足问题规划一样, 决策网络只能进行有限期的规划。

马尔可夫决策过程(MDP)虽然允许不定期或无限期的规划、随机的结果和复杂的偏好, 但采用的却是假定状态是完全可观察的基于状态的表示法。部分可观察马尔可夫决策过程(POMDP)允许存在一个部分可观察的状态, 但求解更为困难。动态决策网络扩展了马尔可夫决策过程, 以允许基于特征的表示法表示状态。动态决策网络也扩展了决策网络, 以允许不定期或无限期的规划, 但无法处理部分可观察状态的情形。

多 Agent 决策网络扩展了决策网络, 以允许存在多个 Agent。策略改进算法允许多个 Agent 进行学习, 但只允许单个状态和规划水平为 1(重复单步游戏); 唯一的不确定性在于其他 Agent 的行为。可以将策略改进算法看成是基于单个状态和多 Agent 的增强学习算法。

增强学习扩展了马尔可夫决策过程, 以允许学习。11.3.9 节讨论了基于特征的增强学习。

情景演算和事件演算允许存在个体和关系的表示以及不定期的规划, 但不能表示不确定性。

独立选择逻辑是一种关系的表示法, 它能表示行为结果的不确定性、感知的不确定性和效用; 然而, 在这个最一般的框架中, 推理并不高效。

构建一个 Agent 最困难的维度是感知的不确定性。多数的表示方法通过激活 Agent 的历史功能(仅对有限期规划有效)来处理感知的不确定性。在部分可观察马尔可夫决策过程中, 讨论了如何扩展基于感知不确定性和不定期及无限期问题的规划, 但其中建议的解决方法依赖于显式的状态。如何处理各种形式的感知问题, 是目前人工智能研究最为活跃的领域之一。

在图 15-2 中, 没有一种方法处理了层次分解问题。有关层次规划和层次增强学习的大部分工作本书并未涉及。

有限理性是许多在实践中获得应用的逼近方法的基础；然而，在思考与行动之间做出明确的折中（Agent 是否立即行动，还是进一步的思考）的研究，相对来讲还是比较少见的。

可以看出，我们只是介绍了人工智能设计空间中的极小一部分内容。目前的研究前沿已经远远超出了本书所涵盖的内容。在人工智能的各个领域都存在许多活跃的研究课题。已经而且将继续在以下各个方面获得引人瞩目的进展：规划、学习、感知器、自然语言理解、机器人以及其他一些子领域。从复杂性维度来看，许多方面的内容是非常深奥的，但依然只是占据了人工智能设计空间的一小部分。

传统上，在关系推理（通常基于一阶逻辑或框架）和不确定性推理（通常基于概率）之间存在一个鸿沟。虽然目前正在填平该鸿沟，但在一些会议和研究论文中依然可见其踪影。

将人工智能研究分解为多个子领域并不令人奇怪。人工智能的设计空间太大了，以致无法同时进行研究。一旦某个研究者决定处理，比如说，关系领域的问题并推导对象是否存在，则增加传感器的不确定性是困难的。如果某个研究者开始进行无限期的规划学习，则增加特征或层次推理是困难的；更不用说基于无限期规划、关系和层次的组合的学习了。

在设计空间中，有一些特殊的研究点在过去的几年里属于前沿研究问题：

- Agent 同时在多个抽象层进行学习的增强学习；
- 多 Agent 增强学习；
- 关系概率学习；
- 考虑歧义、上下文和语用的自然语言理解；
- 能穿过不确定环境的机器人小车；
- 考虑嘈杂传感环境（如：学生情绪）的智能教学系统。

我们仍然不知道如何构建这种关系学习 Agent：针对无限期的规划和存在多个 Agent 的部分可观察域。对这个问题存在一定的争议，有些研究者认为或许可以通过层次近似推理达到该目标。因此，虽然我们还不能构建具有人类水平的智能 Agent，但似乎拥有研制这样一个 Agent 的“积木块”。主要挑战是如何处理现实世界的复杂性。然而，至今似乎不存在构建具有人类水平的计算 Agent 的内在障碍。

15.2 社会与道德后果

随着人工智能科技的发展，出现了越来越多的智能器件，它们的广泛应用会对人类社会及我们的星球产生深刻的道德、心理、社会、经济和法律影响。这里仅概略地说明其中的部分问题。从某种意义上来说，人工自治 Agent 简化了下一阶段的技术发展，这是关于技术开发应用的影响的普遍看法；然而，从另一个方面来讲，它们导致了严重的不连续性。自治 Agent 独立地进行感知、决策和行动。这在我们的技术和技术图景中是一个巨大的质变。这种发展增加了 Agent 挣脱我们的控制进行不可预知行为的可能性。和任意的颠覆性技术一样，可能存在大量的正面和负面结果——对于有些结果，我们将难以评判其好坏；我们也将不能简单地预见某些结果。

为阐述方便，现以大家均熟知的自治车辆作为例子进行说明。实验性自治车辆已经从最初的形式发展到机器人坦克、货物搬运车和自动化战争机器人。从某种意义上讲，机器人战争可能存在显著的优点，但同时又是非常危险的。幸运的是，噩梦仅出现在科幻小说中。

Thrun[2006] 对于这种车辆持乐观态度。拥有智能车辆的正面影响将是巨大的。考虑利用公路进行潜在的生态保护，与直接对农田进行铺路相比，这种方法更高效。在安全方面，智能车辆减少了每年发生在路上的伤亡事故：据估计，全世界每年死于交通事故的人

数达120万人,受伤的人数超过5000万。智能汽车在十字路口能够进行通信和协商。除了减少交通事故,道路的通行量可达到原来的3倍。老人和残障人士可以自己独立地穿行道路。人们可以自主地调度汽车进车库,然后又召回它们。这将真正实现自治汽车的停车自动化,而不是使用地面停车场。在这方面获得成功的正面影响是鼓舞人心的。对于自治车辆发展的后果,存在两种截然不同但却一致的情形:必须考虑其中的道德后果。科幻小说将很快地变成科学事实。

629

无论是作为一门科学,还是作为一门工程学科(技术及其应用),人工智能现在都是发展成熟的。它为我们星球的环境施加正面影响的机会是很多的。人工智能研究人员和开发工程师具备独特的视角和技能;这些视角和技能对于在实践中处理全球变暖、贫穷、粮食生产、军备控制、健康、教育、人口老龄化和人口统计等问题是必需的。作为一个简单的例子,我们可以改善人工智能学习的访问工具,以便增强人们利用人工智能技术解决问题的能力,而非依赖专家构造不透明的系统。正如机器人世界杯足球锦标赛(RoboCup)所获得的成功那样,基于人工智能系统的游戏和竞赛能非常有效地进行学习、教学和环境研究。

我们已经探讨了智能汽车和智能交通控制对于环境方面的一些影响。其他方面的例子如下:已经在频谱分配和物流方面获得应用的组合拍卖技术,可进一步应用于提供碳补偿和优化能源供给与需求;智能节能控制器(使用分布式传感器和激励器,可提高建筑物内部能源利用率)的进一步研究;可以利用定性建模技术对天气状况进行模拟。基于限制的系统的思想可用于分析可持续发展系统。如果一个系统与它所处的环境是平衡的,则称该系统是可持续发展的。“平衡”的意思是系统耗费的资源及其产出不仅满足短期约束,也满足长期约束。

许多研究者开发了服务于残疾人和老年人的辅助技术。辅助认知是一个应用程序,可以辅助感知和行动;它存在的形式包括智能轮椅、老年人伴侣、长期护理设施中的护士助手等。然而,Sharkey[2008]对老年人和小孩依赖机器人助手作为伴侣提出了危险的警告。对于自治车辆,研究人员必须切实回答关于使用其产品的相关问题。

事实上,我们能信任机器人吗?存在一些现实的理由使得我们仍然不能依赖机器人做正确的事情。基于现在构建机器人的方法,它们并非是完全可信赖的。因此,它们能做正确的事情吗?它们愿意做正确的事情吗?什么是正确的事情?在我们的集体潜意识里,存在这样的恐惧:机器人将最终完全自治,具有自由的意志、智能和意识,并可能反叛我们,成为毁灭创造者自己的恶魔。

在人机接口中,道德是什么样的?对于我们自己和机器人,需要道德编码吗?答案看起来是显然的。许多研究者正在研究这个问题。事实上,许多国家已经认识到这是一个重要的有争议的领域。已经存在机器人责任与保险的问题,且将不得不为解决机器人问题而进行立法。机器人设计师和工程师也需要道德的专业编码,就像所有其他学科的工程师需要专业编码一样。我们必须将下述这些问题进行分门别类:在设计、建造和部署机器人的过程中如何考虑道德问题?当机器人发展到更为自治的时候该如何决策?当我们和机器人进行交互的时候,我们如何考虑道德问题和将产生什么道德问题?我们应该赋予机器人什么权利?我们拥有人类权利代码,但存在机器人权利代码吗?

630

我们将上述问题划分为三个基本的必须回答的问题。首先,在设计、建造和部署机器人的过程中,我们人类如何考虑道德问题?第二,当机器人发展到自治和意志自由的时候,它应该怎样符合道德伦理地进行决策?第三,当我们和机器人进行交互的时候,将产生什么样的道德问题?

对于这些问题,我们应考虑科幻小说家 Isaac Asimov[1950]所提出的一些令人感兴趣的建议,虽然这些建议可能不够成熟。Isaac Asimov 是思考这些问题的早期思想家之一。他的机器人三法则是我们理解问题的一个很好的基础,因为它们看起来符合逻辑且简明扼要。最初的三法则是:

- 1) 机器人不得伤害人类,或因不作为使人类受到伤害。
- 2) 除非违背第一法则,机器人必须服从人类的命令。
- 3) 在不违背第一及第二法则的前提下,机器人必须保护自己。

Asimov 对前面提出的三个问题的回答是这样的:首先,每个机器人都必须遵循这些法则,机器人制造商必须依法确保这一点。第二,机器人应该始终遵循法则的优先次序。但是他没有对第三个问题作出更多的说明。Asimov 的观点主要源于人类意欲机器人做什么和机器人真正做了什么之间的冲突;或者是对于上述法则的字面理解和合乎情理的解释之间的差异(因为并没有用任何形式语言将上述法则编成法典)。Asimov 的小说揭示了许多存在于法则和结果之间的隐含矛盾。

现在有许多关于机器人道德伦理的讨论,但大多数都预先假定我们目前还未具备相应的技术能力。事实上, Bill Joy[2000]由于太担心新技术所带来的危险的失控,以至于呼吁暂停研究机器人技术(和人工智能)、纳米技术和基因工程。本书对设计空间作了条理分明的叙述,阐明了包括机器人在内的智能 Agent 的设计原理。我们希望能为智能 Agent 的社会和道德编码的发展提供一个富含技术的框架。

然而,机器人技术可能并非是最具影响的技术。考虑在环球网和其他全球性计算网络中嵌入的、无处不在的分布式智能就可略知一斑。人类和人工智能的混合将演化成为环球智慧。全球性网络对发现和传播新知识的方式的影响已经可以比肩印刷机的发展了。正如 Marshall McLuhan 所说的那样:“我们先是塑造了工具,然后是工具塑造了我们”[McLuhan, 1964]。虽然他说这句话的时候更多的是针对书籍、广告和电视而言,但对于全球性的网络和自治 Agent,它也是适合(甚至是更适合)的。我们已经构建的 Agent 的类型和我们决定构建的 Agent 的类型,将会改变我们自己,并同样多地改变我们所处的社会;我们必须确保它向好的方向发展。Margaret Somerville[2006]是一个伦理学家,他声称:随着以一个不断加速的速率将我们自身的能力固化为技术,物种正在从智人进化为技术人。许多旧的社会和道德框架正在被打破,在这个新的世界中无立足之地。作为人工智能新科技的创立者,我们应该重视新技术所带来的后果,这也是我们共同的责任。

15.3 参考文献及进一步阅读

层次规划的讨论见 Nau[2007]。层次增强学习见 Dietterich[2000]。Stone[2007]讨论了多 Agent 增强学习。

Mackworth[2009]介绍了人工智能带来的某些险境和潜在影响。

Sharkey[2008]和 Singer[2009a, b]概述了机器人战争带来的危险。交通事故的估计数据来源于 Peden[2004],增加交通通行量的估计数据来源于 Dresner and Stone[2008]。

Visser 和 Burkhard[2007]简述了机器人世界杯足球锦标赛(RoboCup)的发展。

辅助技术系统的介绍见 Pollack[2005]以及 Liu、Hile、Kautz、Borriello、Brown、Harniss 和 Johnson[2006],还有 Yang 和 Mackworth[2007]。智能轮椅的讨论见 Mihailidis、Boger、Candido 和 Hoey[2007]以及 Viswanathan、Mackworth、Little 和 Mihailidis[2007]。

Shelley[1818]描写了 Frankenstein 博士和他的怪物。Anderson 和 Leigh Anderson[2007]讨论了机器人道德伦理。Hillis[2008]和 Kelly[2008]引用了环球智慧的概念。

数学基础与记号

本附录介绍人工智能中的一些基础数学概念，这些概念传统上是在其他课程里教授的。也介绍在本书中使用的一些记号和数据结构。

A.1 离散数学

我们所依赖的数学概念包括：

集合 一个集合(set)包含元素(成员)。如果 s 是集合 S 的一个元素，则记为 $s \in S$ 。集合中的元素规定了一个集合，因此，如果两个集合具有相同的元素，则它们是相等的集合。

元组 n -阶元组是有序的 n 个元素的组合，记为 $\langle x_1, \dots, x_n \rangle$ 。2-阶元组是一个对，3-阶元组是 3 个元素的有序组合。如果两个 n -阶元组在相同位置上的元素相同，则称它们是相等的。如果 S 是一个集合，则 S^n 是 n -阶元组 $\langle x_1, \dots, x_n \rangle$ 的集合，其中 x_i 是集合 S 的元素； $S_1 \times S_2 \times \dots \times S_n$ 是 n -阶元组 $\langle x_1, \dots, x_n \rangle$ 的集合，其中 x_i 是集合 S_i 的元素。

关系 关系(relation)是 n -阶元组的集合。关系中的元组称为关系的真值。

函数 一个从定义域(domain) D 到值域(range) R 的函数(function)或映射(mapping) f ，记为 $f: D \rightarrow R$ ，是 $D \times R$ 的一个子集，其中对于任意的 $d \in D$ ，存在一个唯一的 $r \in R$ ，使得 $\langle d, r \rangle \in f$ 。如果 $\langle d, r \rangle \in f$ ，则记为 $f(d) = r$ 。

然而，这些定义可能看起来有些晦涩，如果你对其中的有些概念没有把握，你可以使用相应的常识性知识来加以理解，并检验这些定义。

633

A.2 函数、因子和数组

本书中的许多算法处理函数的表示。我们将定义在集合上的函数扩充到定义在变量集上的函数。**因子**(factor)是函数的一种表示形式。**数组**(array)是函数的一种显式表示，其个体元素可以被修改。

如果 S 是一个集合，则 $f(S)$ 是一个定义域为 S 的函数。因此，如果 $c \in S$ ，则 $f(c)$ 是值域里的一个值。 $f[S]$ 类似于 $f(S)$ ，只不过前者的个体元素可以被更新。这种记号基于 C 语言或 Java 语言中的使用方式(但这些语言仅允许 S 是长度为 n 的整数集合 $\{0, \dots, n-1\}$ 的情形)。因此 $f[c]$ 是值域里的一个值。如果 $f[c]$ 被赋予一个新值，则它将返回该值。

这种记法可以扩展到(代数)变量的情形。如果 X 是一个定义域为 D 的代数变量，则 $f(X)$ 是一个函数，其中，给定一个 $x \in D$ ，返回值域里的一个值。这个值经常被写成 $f(X=x)$ 或简单的 $f(x)$ 。类似的， $f[X]$ 是基于索引 X 的数组，即 X 的函数(元素可以被修改)。

这种记法也可以扩展到变量集的情形。 $f(X_1, X_2, \dots, X_n)$ 是一个函数，其中，给定 X_1 的值 v_1 、 X_2 的值 v_2 、 \dots 、 X_n 的值 v_n ，返回值域里的一个值。注意，这里重要的是变量的名称而非变量的位置。应用于特定值的因子记为 $f(X_1=v_1, X_2=v_2, \dots, X_n=v_n)$ 。变量 X_1, X_2, \dots, X_n 的集合称为 f 的**范围**(scope)。数组 $f[X_1, X_2, \dots, X_n]$ 是 X_1, X_2, \dots, X_n 的函数，其中的值可以被更新。

给一些变量赋值得到一个新的函数，它是其余变量的函数。例如，若 f 是 X_1 ，

X_2, \dots, X_n 的函数, 则 $f(X_1=v_1)$ 是 X_2, \dots, X_n 的函数, 且

$$(f(X_1=v_1))(X_2=v_2, \dots, X_n=v_n) = f(X_1=v_1, X_2=v_2, \dots, X_n=v_n)$$

可以将因子与其他任意的操作算子在元素级别上做相加、相乘或复合。若 f_1 和 f_2 是因子, 则 f_1+f_2 是因子(其范围是 f_1 的范围和 f_2 的范围的并), 定义为

$$(f_1+f_2)(X_1=v_1, X_2=v_2, \dots, X_n=v_n)$$

$$= f_1(X_1=v_1, X_2=v_2, \dots, X_n=v_n) + f_2(X_1=v_1, X_2=v_2, \dots, X_n=v_n)$$

其中假定 f_1 和 f_2 忽略不在范围之内的变量。乘法和其他二元操作的情形与此类似。

【例 A-1】 假定 $f_1(X, Y)=X+Y$ 和 $f_2(Y, Z)=Y+Z$, 则 $f_1+f_2=X+2Y+Z$, 为 X 、 Y 和 Z 的函数。类似的, $f_1 \times f_2=(X+Y) \times (Y+Z)$ 。

634

$f_1(X=2)$ 是 Y 的函数, 定义为 $2+Y$ 。

假定变量 W 的取值范围是 $\{0, 1\}$ 和 X 的取值范围是 $\{1, 2\}$, 则因子 $f_3(W, X)$ 可定义为如下的一个表:

W	X	值
0	1	2
0	2	1
1	1	0
1	2	3

f_3+f_1 是 W 、 X 和 Y 的函数, 例如:

$$(f_3+f_1)(W=1, X=2, Y=3) = 3+5 = 8$$

类似的, $(f_3 \times f_1)(W=1, X=2, Y=3) = 3 \times 5 = 15$

因子上的其他操作的定义可以在本书里找到。

A.3 关系和关系代数

在人工智能和数据库系统中, 关系的概念是常见的。关系代数(relational algebra)定义了关系上的操作, 是关系数据库的基础。

范围(scope) S 是变量的集合。 S 上的元组(tuple) t 对于每个 S 中的变量均有一个值。元组 t 在变量 X 上的值记为 $val(t, X)$, 该值必须在变量 X 的取值范围之内。这和元组的数学记法类似, 只不过是使用变量而不是整数表示索引。

关系是元组的集合, 其中所有的元组具有相同的范围。人们经常为关系给定一个名称。元组的范围经常被称为关系的模式(scheme)。**关系数据库**(relational database)是关系的集合。关系数据库的模式是关系名称和关系模式对的集合。

具有范围 X_1, \dots, X_n 的关系可以看成是在 X_1, \dots, X_n 上的布尔因子, 其中取值为真的元素用元组表示。

一个关系通常被表示成一个表。

【例 A-2】 下面是一个用表描述的关系 *enrolled*:

Course	Year	Student	Grade
cs322	2008	fran	77
cs111	2009	billie	88
cs111	2009	jess	78
cs444	2008	fran	83
cs322	2009	jordan	92

标题行给出了关系的模式, 即 $\{Course, Year, Student, Grade\}$; 其余的每一行均表示一个元组。第一个元组 t_1 定义为 $val(t_1, Course) = cs322$ 、 $val(t_1, Year) = 2008$ 、 $val(t_1, Student) = fran$ 和 $val(t_1, Grade) = 77$ 。

列的顺序和行的顺序是无关紧要的。

若 r 是一个符合模式 S 的关系, c 是定义在 S 中的变量上的一个条件, r 中 c 的选择(selection)是 r 中满足条件 c 的元组的集合, 记为 $\sigma_c(r)$ 。选择具有和 r 相同的模式。

635

若 r 是一个符合模式 S 的关系, $S_0 \subseteq S$, r 在 S_0 上的投影(projection)是范围局限于 S_0 的元组的集合, 记为 $\pi_{S_0}(r)$ 。

【例 A-3】 考虑例 A-2 给出的关系 *enrolled*:

关系 $\sigma_{Grade > 79}(enrolled)$ 选择 *enrolled* 中成绩大于 79 的元组:

Course	Year	Student	Grade
cs111	2009	billie	88
cs444	2008	fran	83
cs322	2009	jordan	92

关系 $\pi_{\{Student, Year\}}(enrolled)$ 在 *enrolled* 中指定对 *Student* 和 *Year* 进行投影:

Student	Year
fran	2008
billie	2009
jess	2009
jordan	2009

注意, 关系 *enrolled* 中的第一个元组和第四个元组是怎样在投影后变为一样的; 它们在 $\{Student, Year\}$ 上表示相同的函数。

如果两个关系的模式相同, 则关系之间的并(union)、交(intersection)和差(difference)定义为元组集上的相应的操作。

若 r_1 和 r_2 是两个关系, 则它们之间的自然连接(join)(记为 $r_1 \bowtie r_2$)是一个关系, 其中:

- 连接的模式是 r_1 的模式和 r_2 的模式的并;
- 一个元组属于连接, 仅当局限于 r_1 的范围的元组属于关系 r_1 和局限于 r_2 的范围的元组属于关系 r_2 。

【例 A-4】 考虑关系 *assisted*:

Course	Year	TA
cs322	2008	yuki
cs111	2009	sam
cs111	2009	chris
cs322	2009	yuki

关系 *enrolled* 和 *assisted* 的连接(记为 $enrolled \bowtie assisted$)为

Course	Year	Student	Grade	TA
cs322	2008	fran	77	yuki
cs111	2009	billie	88	sam
cs111	2009	jess	78	sam
cs111	2009	billie	88	chris
cs111	2009	jess	78	chris
cs322	2009	jordan	92	yuki

注意，在连接中，*cs444* 的信息是如何丢失的：*Course* 为 *cs444* 的元组中不存在 *TA* 的取值。

参考文献

- Aamodt, A. and Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1): 39–59. 341
- Abelson, H. and DiSessa, A. (1981). *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. MIT Press, Cambridge, MA. 66
- Abramson, H. and Rogers, M.H. (Eds.) (1989). *Meta-Programming in Logic Programming*. MIT Press, Cambridge, MA. 592
- Agre, P.E. (1995). Computational research on interaction and agency. *Artificial Intelligence*, 72: 1–52. 66
- Aha, D.W., Marling, C., and Watson, I. (Eds.) (2005). *The Knowledge Engineering Review, special edition on case-based reasoning*, volume 20 (3). Cambridge University Press. 341
- Albus, J.S. (1981). *Brains, Behavior and Robotics*. BYTE Publications, Peterborough, NH. 66
- Allais, M. and Hagen, O. (Eds.) (1979). *Expected Utility Hypothesis and the Allais Paradox*. Reidel, Boston, MA. 380
- Allen, J., Hendler, J., and Tate, A. (Eds.) (1990). *Readings in Planning*. Morgan Kaufmann, San Mateo, CA. 367
- Anderson, M. and Leigh Anderson, S.L. (2007). Machine ethics: Creating an ethical intelligent agent. *AI Magazine*, 28(4): 15–26. 632
- Andrieu, C., de Freitas, N., Doucet, A., and Jordan, M.I. (2003). An introduction to MCMC for machine learning. *Machine Learning*, 50(1–2): 5–43. 275
- Antoniou, G. and van Harmelen, F. (2008). *A Semantic Web Primer*. MIT Press, Cambridge, MA, 2nd edition. 591
- Apt, K. and Bol, R. (1994). Logic programming and negation: A survey. *Journal of Logic Programming*, 19,20: 9–71. 207, 542
- Aristotle (350 B.C.). *Categories*. Translated by E. M. Edghill, <http://www.classicallibrary.org/Aristotle/categories/>. 567
- Asimov, I. (1950). *I, Robot*. Doubleday, Garden City, NY. 631
- Bacchus, F. and Grove, A. (1995). Graphical models for preference and utility. In *Uncertainty in Artificial Intelligence (UAI-95)*, pp. 3–10. 413
- Bacchus, F., Grove, A.J., Halpern, J.Y., and Koller, D. (1996). From statistical knowledge bases to degrees of belief. *Artificial Intelligence*, 87(1–2): 75–143. 274
- Bacchus, F. and Kabanza, F. (1996). Using temporal logic to control search in a forward chaining planner. In M. Ghallab and A. Milani (Eds.), *New Directions in AI Planning*, pp. 141–153. ISO Press, Amsterdam. 367
- Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, NY. 152
- Ballard, B.W. (1983). The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, 21(3): 327–350. 449
- Baum, E.B. (2004). *What is Thought?* MIT Press. 491
- Bayes, T. (1763). An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*, 53: 370–418. Reprinted in *Biometrika* 45, 298–315, 1958. Reprinted in S. J. Press, *Bayesian Statistics*, 189–217, Wiley, New York, 1989. 340

- Beckett, D. and Berners-Lee (2008). Turtle – terse RDF triple language. <http://www.w3.org/TeamSubmission/turtle/>. 592
- Bell, J.L. and Machover, M. (1977). *A Course in Mathematical Logic*. North-Holland, Amsterdam. 207
- Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ. 413
- Bernardo, J.M. and Smith, A.F.M. (1994). *Bayesian Theory*. Wiley. 341
- Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, pp. 28–37. 591
- Bertelè, U. and Brioschi, F. (1972). *Nonserial dynamic programming*, volume 91 of *Mathematics in Science and Engineering*. Academic Press. 151
- Bertsekas, D.P. (1995). *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, Massachusetts. Two volumes. 413
- Bertsekas, D.P. and Tsitsiklis, J.N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, Belmont, Massachusetts. 486
- Besnard, P. and Hunter, A. (2008). *Elements Of Argumentation*. MIT Press, Cambridge, MA. 208
- Bishop, C.M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, England. 341
- Bishop, C.M. (2008). *Pattern Recognition and Machine Learning*. Springer-Verlag, New York. 341
- Blum, A. and Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90: 281–300. 367
- Bobrow, D.G. (1993). Artificial intelligence in perspective: a retrospective on fifty volumes of Artificial Intelligence. *Artificial Intelligence*, 59: 5–20. 41
- Bobrow, D.G. (1967). Natural language input for a computer problem solving system. In M. Minsky (Ed.), *Semantic Information Processing*, pp. 133–215. MIT Press, Cambridge MA. 8
- Boddy, M. and Dean, T.L. (1994). Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2): 245–285. 41
- Bodlaender, H.L. (1993). A tourist guide through treewidth. *Acta Cybernetica*, 11(1-2): 1–21. 274
- Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., and Poole, D. (2004). Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21: 135–191. 413
- Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11: 1–94. 413
- Bowen, K.A. (1985). Meta-level programming and knowledge representation. *New Generation Computing*, 3(4): 359–383. 592
- Bowling, M. and Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2): 215–250. 449
- Brachman, R.J. and Levesque, H.J. (Eds.) (1985). *Readings in Knowledge Representation*. Morgan Kaufmann, San Mateo, CA. 41, 591, 646
- Brachman, R. and Levesque, H. (2004). *Knowledge Representation and Reasoning*. Morgan Kaufmann. 591, 618
- Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA. 341
- Briscoe, G. and Caelli, T. (1996). *A Compendium of Machine Learning, Volume 1: Symbolic Machine Learning*. Ablex, Norwood, NJ. 341

- Brooks, R.A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1): 14–23. Reprinted in Shafer and Pearl [1990]. 66
- Brooks, R.A. (1991). Intelligence without representation. *Artificial Intelligence*, 47: 139–159. 66
- Brooks, R. (1990). Elephants don't play chess. *Robotics and Autonomous Systems*, 6: 3–15. 41
- Bryce, D. and Kambhampati, S. (2007). A tutorial on planning graph based reachability heuristics. *AI Magazine*, 28(47–83): 1. 367
- Buchanan, B. and Shortliffe, E. (Eds.) (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, MA. 9
- Buchanan, B.G. and Feigenbaum, E.A. (1978). Dendral and meta-dendral: Their applications dimension. *Artificial Intelligence*, 11: 5–24. 9
- Buchanan, B.G. (2005). A (very) brief history of artificial intelligence. *AI Magazine*, 26(4): 53–60. 3
- Buntine, W. (1992). Learning classification trees. *Statistics and Computing*, 2: 63–73. 341
- Burch, R. (2008). Charles Sanders Peirce. *The Stanford Encyclopedia of Philosophy*. <http://plato.stanford.edu/archives/spr2008/entries/peirce/>. 207
- Campbell, M., Hoane Jr., A.J., and Hse, F.h. (2002). Deep blue. *Artificial Intelligence*, 134(1–2): 57–83. 449
- Castillo, E., Gutiérrez, J.M., and Hadi, A.S. (1996). *Expert Systems and Probabilistic Network Models*. Springer Verlag, New York. 274
- Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, 32(3): 333–377. 367
- Cheeseman, P. (1990). On finding the most probable model. In J. Shragner and P. Langley (Eds.), *Computational Models of Scientific Discovery and Theory Formation*, chapter 3, pp. 73–95. Morgan Kaufmann, San Mateo, CA. 341
- Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., and Freeman, D. (1988). Autoclass: A Bayesian classification system. In *Proc. Fifth International Conference on Machine Learning*, pp. 54–64. Ann Arbor, MI. Reprinted in Shavlik and Dietterich [1990]. 486
- Cheng, J. and Druzdzel, M. (2000). AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Journal of Artificial Intelligence Research*, 13: 155–188. 275
- Chesnevar, C., Maguitman, A., and Loui, R. (2000). Logical models of argument. *ACM Comput. Surv.*, 32(4): 337–383. 208
- Chomsky, N. (1957). *Syntactic Structures*. Mouton and Co., The Hague. 521
- Chrisley, R. and Begeer, S. (2000). *Artificial intelligence: Critical Concepts in Cognitive Science*. Routledge, London and New York. 41, 645
- Clark, K.L. (1978). Negation as failure. In H. Gallaire and J. Minker (Eds.), *Logic and Databases*, pp. 293–322. Plenum Press, New York. 207, 542
- Cohen, P.R. (2005). If not Turing's test, then what? *AI Magazine*, 26(4): 61–67. 40
- Colmerauer, A., Kanoui, H., Roussel, P., and Pasero, R. (1973). Un système de communication homme-machine en français. Technical report, Groupe de Recherche en Intelligence Artificielle, Université d'Aix-Marseille. 207
- Colmerauer, A. and Roussel, P. (1996). The birth of Prolog. In T.J. Bergin and R.G. Gibson (Eds.), *History of Programming Languages*. ACM Press/Addison-Wesley. 9, 542
- Copi, I.M. (1982). *Introduction to Logic*. Macmillan, New York, sixth edition. 207
- Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition. 106

- Cover, T.M. and Thomas, J.A. (1991). *Elements of information theory*. Wiley, New York. 275
- Culberson, J. and Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, 14(3): 318–334. 106
- Dahl, V. (1994). Natural language processing and logic programming. *Journal of Logic Programming*, 19,20: 681–714. 542
- Darwiche, A. (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press. 274, 486
- Dasarathy, B.V. (1991). NN concepts and techniques. In B.V. Dasarathy (Ed.), *Nearest Neighbour (NN) Norms: NN Pattern Classification Techniques*, pp. 1–30. IEEE Computer Society Press, New York. 341
- Davis, E. (1990). *Representations of Commonsense Knowledge*. Morgan Kaufmann, San Mateo, CA. 591
- Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine Learning*, pp. 233–240. 341
- Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem proving. *Communications of the ACM*, 5(7): 394–397. 151
- Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, 7(3): 201–215. 151
- de Kleer, J. (1986). An assumption-based TMS. *Artificial Intelligence*, 28(2): 127–162. 207, 208
- de Kleer, J., Mackworth, A.K., and Reiter, R. (1992). Characterizing diagnoses and systems. *Artificial Intelligence*, 56: 197–222. 207
- De Raedt, L., Frasconi, P., Kersting, K., and Muggleton, S.H. (Eds.) (2008). *Probabilistic Inductive Logic Programming*. Springer. 620
- Dean, T. and Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3): 142–150. 275
- Dean, T.L. and Wellman, M.P. (1991). *Planning and Control*. Morgan Kaufmann, San Mateo, CA. 41, 66
- Dechter, R. (1996). Bucket elimination: A unifying framework for probabilistic inference. In E. Horvitz and F. Jensen (Eds.), *Proc. Twelfth Conf. on Uncertainty in Artificial Intelligence (UAI-96)*, pp. 211–219. Portland, OR. 274
- Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann. 151
- Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999). Monte Carlo localization for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA99)*. 275
- Dietterich, T.G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13: 227–303. 632
- Dietterich, T.G. (2002). Ensemble learning. In M. Arbib (Ed.), *The Handbook of Brain Theory and Neural Networks*, pp. 405–408. MIT Press, Cambridge, MA, second edition. 341
- Dijkstra, E.W. (1976). *A discipline of programming*. Prentice-Hall, Englewood Cliffs, NJ. 367
- Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1: 269–271. 106
- Doucet, A., de Freitas, N., and Gordon, N. (Eds.) (2001). *Sequential Monte Carlo in Practice*. Springer-Verlag. 275
- Doyle, J. (1979). A truth maintenance system. AI Memo 521, MIT Artificial Intelligence Laboratory. 207
- Dresner, K. and Stone, P. (2008). A multiagent approach to autonomous intersection

- management. *Journal of Artificial Intelligence Research*, 31: 591–656. 632
- Duda, R.O., Hart, P.E., and Stork, D.G. (2001). *Pattern Classification*. Wiley-Interscience, 2nd edition. 341, 486
- Dung, P. (1995). On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n -person games. *Artificial Intelligence*, 77(2): 321–357. 207
- Dung, P., Mancarella, P., and Toni, F. (2007). Computing ideal sceptical argumentation. *Artificial Intelligence*, 171(10–15): 642–674. 207
- Edwards, P. (Ed.) (1967). *The Encyclopedia of Philosophy*. Macmillan, New York. 287
- Enderton, H.B. (1972). *A Mathematical Introduction to Logic*. Academic Press, Orlando, FL. 207
- Felner, A., Korf, R.E., and Hanan, S. (2004). Additive pattern database heuristics. *Journal of Artificial Intelligence Research (JAIR)*, 22: 279–318. 106
- Fikes, R.E. and Nilsson, N.J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4): 189–208. 367
- Fischer, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2: 139–172. Reprinted in Shavlik and Dietterich [1990]. 486
- Forbus, K.D. (1996). Qualitative reasoning. In *CRC Hand-book of Computer Science and Engineering*. CRC Press. 66
- Freuder, E.C. and Mackworth, A.K. (2006). Constraint satisfaction: An emerging paradigm. In P.V.B. F. Rossi and T. Walsh (Eds.), *Handbook of Constraint Programming*, pp. 13–28. Elsevier. 151
- Friedman, N. and Goldszmidt, M. (1996a). Building classifiers using Bayesian networks. In *Proc. 13th National Conference on Artificial Intelligence*, pp. 1277–1284. Portland, OR. 486
- Friedman, N. and Goldszmidt, M. (1996b). Learning Bayesian networks with local structure. In *Proc. Twelfth Conf. on Uncertainty in Artificial Intelligence (UAI-96)*, pp. 252–262. 486
- Friedman, N., Greiger, D., and Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, 29: 103–130. 341
- Gabbay, D.M., Hogger, C.J., and Robinson, J.A. (Eds.) (1993). *Handbook of Logic in Artificial Intelligence and Logic Programming*. Clarendon Press, Oxford, England. 5 volumes. 207
- Gangemi, A., Guarino, N., Masolo, C., and Oltramari, A. (2003). Sweetening wordnet with dolce. *AI Magazine*, 24(3): 13–24. 592
- Garcia-Molina, H., Ullman, J.D., and Widom, J. (2009). *Database Systems: The Complete Book*. Prentice Hall, 2nd edition. 542
- Gardner, H. (1985). *The Mind's New Science*. Basic Books, New York. 41
- Gelman, A., Carlin, J.B., Stern, H.S., and Rubin, D.B. (2004). *Bayesian Data Analysis*. Chapman and Hall/CRC, 2nd edition. 341
- Getoor, L. and Taskar, B. (Eds.) (2007). *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA. 620
- Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA. 152
- Goldberg, D.E. (2002). *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Addison-Wesley, Reading, MA. 152
- Green, C. (1969). Application of theorem proving to problem solving. In *Proc. 1st International Joint Conf. on Artificial Intelligence*, pp. 219–237. Washington, DC. Reprinted in Webber and Nilsson [1981]. 207
- Grenon, P. and Smith, B. (2004). Snap and span: Towards dynamic spatial ontology. *Spatial Cognition and Computation*, 4(1): 69–103. 592

- Grünwald, P.D. (2007). *The Minimum Description Length Principle*. The MIT Press, Cambridge, MA. 275, 341
- Halpern, J. (1997). A logical approach to reasoning about uncertainty: A tutorial. In X. Arrazola, K. Kortha, and F. Pelletier (Eds.), *Discourse, Interaction and Communication*. Kluwer. 274
- Hart, P.E., Nilsson, N.J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107. 106
- Hart, T.P. and Edwards, D.J. (1961). The tree prune (TP) algorithm. Memo 30, MIT Artificial Intelligence Project, Cambridge MA. 449
- Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, second edition. 341
- Haugeland, J. (1985). *Artificial Intelligence: The Very Idea*. MIT Press, Cambridge, MA. 6, 41, 66
- Haugeland, J. (Ed.) (1997). *Mind Design II: Philosophy, Psychology, Artificial Intelligence*. MIT Press, Cambridge, MA, revised and enlarged edition. 40, 646, 647, 651
- Haussler, D. (1988). Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36(2): 177–221. Reprinted in Shavlik and Dietterich [1990]. 341
- Hayes, P.J. (1973). Computation and deduction. In *Proc. 2nd Symposium on Mathematical Foundations of Computer Science*, pp. 105–118. Czechoslovak Academy of Sciences. 207
- Heckerman, D. (1999). A tutorial on learning with Bayesian networks. In M. Jordan (Ed.), *Learning in Graphical Models*. MIT press. 486
- Hendler, J., Berners-Lee, T., and Miller, E. (2002). Integrating applications on the semantic web. *Journal of the Institute of Electrical Engineers of Japan*, 122(10): 676–680. 591
- Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In J.F. Lemmer and L.N. Kanal (Eds.), *Uncertainty in Artificial Intelligence 2*, pp. 149–163. Elsevier Science Publishers B.V. 275
- Hertz, J., Krogh, A., and Palmer, R.G. (1991). *Introduction to the Theory of Neural Computation*. Lecture Notes, Volume I, Santa Fe Institute Studies in the Sciences of Complexity. Addison-Wesley, Reading, MA. 341
- Hewitt, C. (1969). Planner: A language for proving theorems in robots. In *Proc. 1st International Joint Conf. on Artificial Intelligence*, pp. 295–301. Washington, DC. 207
- Hillis, W.D. (2008). A forebrain for the world mind. *Edge: World Question Center*. http://www.edge.org/q2009/q09_12.html#hillis. 632
- Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., and Rudolph, S. (2009). *OWL 2 Web Ontology Language Primer*. W3C. <http://www.w3.org/TR/owl2-primer/>. 592
- Hobbs, J.R., Stickel, M.E., Appelt, D.E., and Martin, P. (1993). Interpretation as abduction. *Artificial Intelligence*, 63(1–2): 69–142. 207
- Holland, J.H. (1975). *Adaption in Natural and Artificial Systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, Ann Arbor, MI. 152
- Hoos, H.H. and Stützle, T. (2004). *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann / Elsevier. 152
- Horvitz, E.J. (1989). Reasoning about beliefs and actions under computational resource constraints. In L. Kanal, T. Levitt, and J. Lemmer (Eds.), *Uncertainty in Artificial Intelligence 3*, pp. 301–324. Elsevier, New York. 41
- Horvitz, E. (2006). Eric Horvitz forecasts the future. *New Scientist*, 2578: 72. 625
- Howard, R.A. and Matheson, J.E. (1984). Influence diagrams. In R.A. Howard and J.E. Matheson (Eds.), *The Principles and Applications of Decision Analysis*. Strategic Deci-

- sions Group, Menlo Park, CA. 413
- Howson, C. and Urbach, P. (2006). *Scientific Reasoning: the Bayesian Approach*. Open Court, Chicago, Illinois, 3rd edition. 341
- Jaynes, E.T. (2003). *Probability Theory: The Logic of Science*. Cambridge University Press. 341
- Jensen, F.V. (1996). *An Introduction to Bayesian Networks*. Springer Verlag, New York. 274
- Jordan, M. and Bishop, C. (1996). Neural networks. Memo 1562, MIT Artificial Intelligence Lab, Cambridge, MA. 341
- Joy, B. (2000). Why the future doesn't need us. *Wired*. <http://www.wired.com/wired/archive/8.04/joy.html>. 631
- Jurafsky, D. and Martin, J.H. (2008). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, second edition. 542
- Kaelbling, L.P., Littman, M.L., and Moore, A.W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4: 237–285. 486
- Kakas, A. and Denecker, M. (2002). Abduction in logic programming. In A. Kakas and F. Sadri (Eds.), *Computational Logic: Logic Programming and Beyond*, number 2407 in LNAI, pp. 402–436. Springer Verlag. 207
- Kakas, A.C., Kowalski, R.A., and Toni, F. (1993). Abductive logic programming. *Journal of Logic and Computation*, 2(6): 719–770. 207
- Kambhampati, S., Knoblock, C.A., and Yang, Q. (1995). Planning as refinement search: a unified framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence*, 76: 167–238. Special issue on Planning and Scheduling. 367
- Kautz, H. and Selman, B. (1996). Pushing the envelope: Planning, propositional logic and stochastic search. In *Proc. 13th National Conference on Artificial Intelligence*, pp. 1194–1201. Portland, OR. 367
- Kearns, M. and Vazirani, U. (1994). *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA. 341
- Keeney, R.L. and Raiffa, H. (1976). *Decisions with Multiple Objectives*. John Wiley and Sons. 413
- Kelly, K. (2008). A new kind of mind. *Edge: World Question Center*. <http://www.edge.org/q2009/q09.1.html#kelly>. 632
- Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1983). Optimization by simulated annealing. *Science*, 220: 671–680. 152
- Kirsh, D. (1991a). Foundations of AI: the big issues. *Artificial Intelligence*, 47: 3–30. 41
- Kirsh, D. (1991b). Today the earwig, tomorrow man? *Artificial Intelligence*, 47: 161–184. 66
- Knuth, D.E. and Moore, R.W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4): 293–326. 449
- Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press. 274, 486
- Koller, D. and Milch, B. (2003). Multi-agent influence diagrams for representing and solving games. *Games and Economic Behavior*, 45(1): 181–221. 449
- Kolodner, J. and Leake, D. (1996). A tutorial introduction to case-based reasoning. In D. Leake (Ed.), *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, pp. 31–65. AAAI Press/MIT Press. 341
- Korf, K.E. (1985). Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1): 97–109. 106
- Kowalski, R. (1979). *Logic for Problem Solving*. Artificial Intelligence Series. North-Holland, New York. 542, 592, 618

- Kowalski, R. and Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, 4(1): 67–95. 620
- Kowalski, R.A. (1974). Predicate logic as a programming language. In *Information Processing 74*, pp. 569–574. North-Holland, Stockholm. 207
- Kowalski, R.A. (1988). The early history of logic programming. *CACM*, 31(1): 38–43. 9, 542
- Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA. 152
- Kuipers, B. (2001). Qualitative simulation. In R.A. Meyers (Ed.), *Encyclopedia of Physical Science and Technology*, pp. 287–300. Academic Press, NY, third edition. 66
- Langley, P., Iba, W., and Thompson, K. (1992). An analysis of Bayesian classifiers. In *Proc. 10th National Conference on Artificial Intelligence*, pp. 223–228. San Jose, CA. 486
- Laplace, P. (1812). *Théorie Analytique de Probabilités*. Courcier, Paris. 219, 223, 296
- Latombe, J.C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers, Boston. 66
- Lawler, E.L. and Wood, D.E. (1966). Branch-and-bound methods: A survey. *Operations Research*, 14(4): 699–719. 106
- Leibniz, G.W. (1677). *The Method of Mathematics: Preface to the General Science*. Selections reprinted by Chrisley and Begeer [2000]. 157
- Lenat, D.B. and Feigenbaum, E.A. (1991). On the thresholds of knowledge. *Artificial Intelligence*, 47: 185–250. 41
- Levesque, H.J. (1984). Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, 23(2): 155–212. 207
- Liu, A.L., Hile, H., Kautz, H., Borriello, G., Brown, P.A., Harniss, M., and Johnson, K. (2006). Indoor wayfinding: Developing a functional interface for individuals with cognitive impairments. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility*, pp. 95–102. Association for Computing Machinery, New York. 632
- Lloyd, J.W. (1987). *Foundations of Logic Programming*. Symbolic Computation Series. Springer-Verlag, Berlin, second edition. 207, 542
- Lopez, A. and Bacchus, F. (2003). Generalizing GraphPlan by formulating planning as a CSP. In *IJCAI-03*, pp. 954–960. 367
- Lopez De Mantaras, R., Mcsherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M.L., Cox, M.T., Forbus, K., Keane, M., Aamodt, A., and Watson, I. (2005). Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review*, 20(3): 215–240. 341
- Loredo, T. (1990). From Laplace to supernova SN 1987A: Bayesian inference in astrophysics. In P. Fougère (Ed.), *Maximum Entropy and Bayesian Methods*, pp. 81–142. Kluwer Academic Press, Dordrecht, The Netherlands. 341
- Lowe, D.G. (1995). Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 7: 72–85. 341
- Luenberger, D.G. (1979). *Introduction to Dynamic Systems: Theory, Models and Applications*. Wiley, New York. 66
- Mackworth, A.K. (1993). On seeing robots. In A. Basu and X. Li (Eds.), *Computer Vision: Systems, Theory, and Applications*, pp. 1–13. World Scientific Press, Singapore. 66
- Mackworth, A.K. (2009). Agents, bodies, constraints, dynamics and evolution. *AI Magazine*. 632
- Mackworth, A.K. (1977). On reading sketch maps. In *Proc. Fifth International Joint Conf. on Artificial Intelligence*, pp. 598–606. MIT, Cambridge, MA. 151
- Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA. 542
- Mas-Colell, A., Whinston, M.D., and Green, J.R. (1995). *Microeconomic Theory*. Oxford

- University Press, New York, NY. 449
- Matheson, J.E. (1990). Using influence diagrams to value information and control. In R.M. Oliver and J.Q. Smith (Eds.), *Influence Diagrams, Belief Nets and Decision Analysis*, chapter 1, pp. 25–48. Wiley. 413
- McAllester, D. and Rosenblitt, D. (1991). Systematic nonlinear planning. In *Proc. 9th National Conference on Artificial Intelligence*, pp. 634–639. 367
- McCarthy, J. (1986). Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28(1): 89–116. 207
- McCarthy, J. and Hayes, P.J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In M. Meltzer and D. Michie (Eds.), *Machine Intelligence 4*, pp. 463–502. Edinburgh University Press. 8, 618
- McCulloch, W. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5: 115–133. 7
- McDermott, D. and Hendler, J. (1995). Planning: What it is, what it could be, an introduction to the special issue on planning and scheduling. *Artificial Intelligence*, 76: 1–16. 367
- McLuhan, M. (1964). *Understanding Media: The Extensions of Man*. New American Library, New York. 632
- Meir, R. and Rätsch, G. (2003). An introduction to boosting and leveraging. In *Advanced Lectures on Machine Learning (LNAI2600)*, pp. 119–184. Springer. 341
- Mendelson, E. (1987). *Introduction to Mathematical Logic*. Wadsworth and Brooks, Monterey, CA, third edition. 207
- Michie, D., Spiegelhalter, D.J., and Taylor, C.C. (Eds.) (1994). *Machine Learning, Neural and Statistical Classification*. Series in Artificial Intelligence. Ellis Horwood, Hemel Hempstead, Hertfordshire, England. 341
- Mihailidis, A., Boger, J., Candido, M., and Hoey, J. (2007). The use of an intelligent prompting system for people with dementia. *ACM Interactions*, 14(4): 34–37. 632
- Minsky, M. (1961). Steps towards artificial intelligence. *Proceedings of the IEEE*, 49: 8–30. 106
- Minsky, M. (1986). *The Society of Mind*. Simon and Schuster, New York. 29
- Minsky, M. and Papert, S. (1988). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, expanded edition. 7, 341
- Minsky, M.L. (1975). A framework for representing knowledge. In P. Winston (Ed.), *The Psychology of Computer Vision*, pp. 211–277. McGraw-Hill, New York. Alternative version is in Haugeland [1997], and reprinted in Brachman and Levesque [1985]. 8
- Minsky, M. (1952). A neural-analogue calculator based upon a probability model of reinforcement. Technical report, Harvard University Psychological Laboratories, Cambridge, MA. 7
- Minton, S., Johnston, M.D., Philips, A.B., and Laird, P. (1992). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3): 161–205. 152
- Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA. 152
- Mitchell, T. (1997). *Machine Learning*. McGraw-Hill, New York. 341
- Mitchell, T.M. (1977). Version spaces: A candidate elimination approach to rule learning. In *Proc. 5th International Joint Conf. on Artificial Intelligence*, pp. 305–310. Cambridge, MA. 341
- Motik, B., Patel-Schneider, P.F., and Grau, B.C. (Eds.) (2009a). *OWL 2 Web Ontology Language Direct Semantics*. W3C. <http://www.w3.org/TR/owl2-semantics/>. 592
- Motik, B., Patel-Schneider, P.F., and Parsia, B. (Eds.) (2009b). *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax*. W3C. <http://www>

- .w3.org/TR/owl2-syntax/. 592
- Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, 13(3/4): 245–286. 620
- Muggleton, S. and De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20: 629–679. 620
- Muscettola, N., Nayak, P., Pell, B., and Williams, B. (1998). Remote agent: to boldly go where no AI system has gone before. *Artificial Intelligence*, 103: 5–47. 212
- Nash, Jr., J.F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36: 48–49. 436
- Nau, D.S. (2007). Current trends in automated planning. *AI Magazine*, 28(4): 43–58. 367, 632
- Neumann, J.V. and Morgenstern, O. (1953). *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ, third edition. 413, 424
- Newell, A. and Simon, H.A. (1976). Computer science as empirical enquiry: Symbols and search. *Communications of the ACM*, 19: 113–126. Reprinted in Haugeland [1997]. 15, 41
- Newell, A. and Simon, H.A. (1956). The logic theory machine: A complex information processing system. Technical Report P-868, The Rand Corporation. 7
- Niles, I. and Pease, A. (2001). Towards a standard upper ontology. In C. Welty and B. Smith (Eds.), *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*. Ogunquit, Maine. 592
- Nilsson, N. (2007). The physical symbol system hypothesis: Status and prospects. In e.a. M. Lungarella (Ed.), *50 Years of AI, Festschrift*, volume 4850 of *LNAI*, pp. 9–17. Springer. 41
- Nilsson, N.J. (1971). *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York. 106
- Nilsson, N.J. (2009). *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge University Press, Cambridge, England. 40
- Nisan, N. (2007). Introduction to mechanism design (for computer scientists). In N.N. et al. (Ed.), *Algorithmic Game Theory*, chapter 9, pp. 209–242. Cambridge University Press, Cambridge, England. 449
- Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V.V. (Eds.) (2007). *Algorithmic Game Theory*. Cambridge University Press. 449
- Noy, N.F. and Hafner, C.D. (1997). The state of the art in ontology design: A survey and comparative review. *AI Magazine*, 18(3): 53–74. 592
- Ordeshook, P.C. (1986). *Game theory and political theory: An introduction*. Cambridge University Press, New York. 449
- Panton, K., Matuszek, C., Lenat, D., Schneider, D., Witbrock, M., Siegel, N., and Shepard, B. (2006). Common sense reasoning – from Cyc to intelligent assistant. In Y. Cai and J. Abascal (Eds.), *Ambient Intelligence in Everyday Life*, *LNAI* 3864, pp. 1–31. Springer. 592
- Pearl, J. (1984). *Heuristics*. Addison-Wesley, Reading, MA. 106, 449
- Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA. 274
- Pearl, J. (2000). *Causality: Models, Reasoning and Inference*. Cambridge University Press. 208, 275
- Peden, M.e.a. (Ed.) (2004). *World Report on Road Traffic Injury Prevention*. World Health Organization, Geneva. 632
- Peng, Y. and Reggia, J.A. (1990). *Abductive Inference Models for Diagnostic Problem-Solving*. Symbolic Computation – AI Series. Springer-Verlag, New York. 207

- Pereira, F.C.N. and Shieber, S.M. (2002). *Prolog and Natural-Language Analysis*. Microtome Publishing. 542
- Pollack, M.E. (2005). Intelligent technology for an aging population: The use of ai to assist elders with cognitive impairment. *AI Magazine*, 26(2): 9–24. 632
- Poole, D. (1993). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1): 81–129. 620
- Poole, D. (1997). The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94: 7–56. Special issue on economic principles of multi-agent systems. 620
- Poole, D., Goebel, R., and Aleliunas, R. (1987). Theorist: A logical reasoning system for defaults and diagnosis. In N. Cercone and G. McCalla (Eds.), *The Knowledge Frontier: Essays in the Representation of Knowledge*, pp. 331–352. Springer-Verlag, New York, NY. 207
- Poole, D., Mackworth, A., and Goebel, R. (1998). *Computational Intelligence: A Logical Approach*. Oxford University Press, New York. xiv
- Posner, M.I. (Ed.) (1989). *Foundations of Cognitive Science*. MIT Press, Cambridge, MA. 41
- Price, C.J., Travé-Massuyàs, L., Milne, R., Ironi, L., Forbus, K., Bredeweg, B., Lee, M.H., Struss, P., Snooke, N., Lucas, P., Cavazza, M., and Coghill, G.M. (2006). Qualitative futures. *The Knowledge Engineering Review*, 21(04): 317–334. 66
- Puterman, M. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, New York. 413
- Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning*, 1: 81–106. Reprinted in Shavlik and Dietterich [1990]. 341
- Quinlan, J.R. (1993). *C4.5 Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA. 341
- Quinlan, J.R. and Cameron-Jones, R.M. (1995). Induction of logic programs: FOIL and related systems. *New Generation Computing*, 13(3,4): 287–312. 620
- Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2): 257–286. 275
- Reiter, R. (1991). The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz (Ed.), *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pp. 359–380. Academic Press, San Diego, CA. 618
- Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press. 618
- Riesbeck, C. and Schank, R. (1989). *Inside Case-Based Reasoning*. Lawrence Erlbaum, Hillsdale, NJ. 341
- Robinson, J.A. (1965). A machine-oriented logic based on the resolution principle. *Journal ACM*, 12(1): 23–41. 207
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6): 386–408. 7, 306
- Rosenschein, S.J. and Kaelbling, L.P. (1995). A situated view of representation and control. *Artificial Intelligence*, 73: 149–173. 66
- Rubinstein, R.Y. (1981). *Simulation and the Monte Carlo Method*. John Wiley and Sons. 275
- Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986). Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland (Eds.), *Parallel Distributed Processing*, chapter 8, pp. 318–362. MIT Press, Cambridge, MA. Reprinted in Shavlik and Dietterich [1990]. 341
- Russell, B. (1917). *Mysticism and Logic and Other Essays*. G. Allen and Unwin, London.

597

- Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Series in Artificial Intelligence. Prentice-Hall, Englewood Cliffs, NJ, third edition. 41
- Russell, S. (1997). Rationality and intelligence. *Artificial Intelligence*, 94: 57–77. 41
- Sacerdoti, E.D. (1975). The nonlinear nature of plans. In *Proc. 4th International Joint Conf. on Artificial Intelligence*, pp. 206–214. Tbilisi, Georgia, USSR. 367
- Samuel, A.L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3(3): 210–229. 7, 424
- Sandholm, T. (2007). Expressive commerce and its application to sourcing: How we conducted \$35 billion of generalized combinatorial auctions. *AI Magazine*, 28(3): 45–58. 41
- Savage, L.J. (1972). *The Foundation of Statistics*. Dover, New York, 2nd edition. 413
- Schank, R.C. (1990). What is AI, anyway? In D. Partridge and Y. Wilks (Eds.), *The Foundations of Artificial Intelligence*, pp. 3–13. Cambridge University Press, Cambridge, England. 41
- Schapire, R.E. (2002). The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*. Springer Verlag. 341
- Schubert, L.K. (1990). Monotonic solutions to the frame problem in the situation calculus: An efficient method for worlds with fully specified actions. In H.E. Kyburg, R.P. Loui, and G.N. Carlson (Eds.), *Knowledge Representation and Defeasible Reasoning*, pp. 23–67. Kluwer Academic Press, Boston, MA. 618
- Shachter, R. and Peot, M.A. (1992). Decision making using probabilistic inference methods. In *Proc. Eighth Conf. on Uncertainty in Artificial Intelligence (UAI-92)*, pp. 276–283. Stanford, CA. 413
- Shafer, G. and Pearl, J. (Eds.) (1990). *Readings in Uncertain Reasoning*. Morgan Kaufmann, San Mateo, CA. 639
- Shanahan, M. (1989). Prediction is deduction, but explanation is abduction. In *Proc. 11th International Joint Conf. on Artificial Intelligence (IJCAI-89)*, pp. 1055–1060. Detroit, MI. 207
- Shanahan, M. (1997). *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, Cambridge, MA. 620
- Shapiro, S.C. (Ed.) (1992). *Encyclopedia of Artificial Intelligence*. Wiley, New York, second edition. 41
- Sharkey, N. (2008). The ethical frontiers of robotics. *Science*, 322(5909): 1800–1801. 630, 632
- Shavlik, J.W. and Dietterich, T.G. (Eds.) (1990). *Readings in Machine Learning*. Morgan Kaufmann, San Mateo, CA. 341, 640, 642, 643, 648, 649, 651
- Shelley, M.W. (1818). *Frankenstein; or, The Modern Prometheus*. Lackington, Hughes, Harding, Mavor and Jones, London. 632
- Shoham, Y. and Leyton-Brown, K. (2008). *Multiagent Systems: Algorithmic, Game Theoretic, and Logical Foundations*. Cambridge University Press. 423, 449
- Simon, H.A. (1995). Artificial intelligence: an empirical science. *Artificial Intelligence*, 77(1): 95–127. 41
- Simon, H. (1996). *The Sciences of the Artificial*. MIT Press, Cambridge, MA, third edition. 41, 43, 66
- Singer, P.W. (2009a). Robots at war: The new battlefield. *The Wilson Quarterly*. <http://www.wilsoncenter.org/index.cfm?fuseaction=wq.essay&essay.id=496613>. 632
- Singer, P.W. (2009b). *Wired for War: The Robotics Revolution and Conflict in the 21st Century*. Penguin, New York. 632
- Smith, B. (2003). Ontology. In L. Floridi (Ed.), *Blackwell Guide to the Philosophy of Computing and Information*, pp. 155–166. Oxford: Blackwell. 591
- Smith, B.C. (1991). The owl and the electric encyclopedia. *Artificial Intelligence*, 47: 251–

288. 41
- Smith, B.C. (1996). *On the Origin of Objects*. MIT Press, Cambridge, MA. 549
- Somerville, M. (2006). *The Ethical Imagination: Journeys of the Human Spirit*. House of Anansi Press, Toronto. 632
- Spall, J.C. (2003). *Introduction to Stochastic Search and Optimization: Estimation, Simulation*. Wiley. 152
- Spiegelhalter, D.J., Franklin, R.C.G., and Bull, K. (1990). Assessment, criticism and improvement of imprecise subjective probabilities for a medical expert system. In M. Henrion, R.D. Shachter, L. Kanal, and J. Lemmer (Eds.), *Uncertainty in Artificial Intelligence 5*, pp. 285–294. North-Holland, Amsterdam, The Netherlands. 341
- Spirtes, P., Glymour, C., and Scheines, R. (2000). *Causation, Prediction, and Search*. MIT Press, Cambridge MA, 2nd edition. 208, 275
- Sterling, L. and Shapiro, E. (1986). *The Art of Prolog*. MIT Press, Cambridge, MA. 542
- Stillings, N.A., Feinstein, M.H., Garfield, J.L., Rissland, E.L., Rosenbaum, D.A., Weisler, S.E., and Baker-Ward, L. (1987). *Cognitive Science: An Introduction*. MIT Press, Cambridge, MA. 41
- Stone, P. (2007). Learning and multiagent reasoning for autonomous agents. In *The 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pp. 13–30. 632
- Stone, P. and Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8: 345–383. 449
- Sutton, R.S. and Barto, A.G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA. 486
- Tarski, A. (1956). *Logic, Semantics, Metamathematics*. Clarendon Press, Oxford, England. Papers from 1923 to 1938 collected and translated by J. H. Woodger. 207
- Tate, A. (1977). Generating project networks. In *Proc. 5th International Joint Conf. on Artificial Intelligence*, pp. 888–893. Cambridge, MA. 367
- Tharp, T. (2003). *The Creative Habit: Learn It and Use It for Life*. Simon and Schuster. 371
- Thrun, S. (2006). Winning the darpa grand challenge. In *Innovative Applications of Artificial Intelligence Conference, (IAAI-06)*, pp. 16–20. Boston, MA. 629
- Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press, Cambridge, MA. 275
- Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59: 433–460. Reprinted in Haugeland [1997]. 5, 40
- Tversky, A. and Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. *Science*, 185: 1124–1131. 380
- Valiant, L.G. (1984). A theory of the learnable. *Communications of the ACM*, 27: 1134–1142. Reprinted in Shavlik and Dietterich [1990]. 341
- van Beek, P. and Chen, X. (1999). Cplan: A constraint programming approach to planning. In *AAAI-99*, pp. 585–590. 367
- van Emden, M.H. and Kowalski, R.A. (1976). The semantics of predicate logic as a programming language. *Journal ACM*, 23(4): 733–742. 207
- Visser, U. and Burkhard, H.D. (2007). Robocup: 10 years of achievements and challenges. *AI Magazine*, 28(2): 115–130. 632
- Viswanathan, P., Mackworth, A.K., Little, J.J., and Mihailidis, A. (2007). Intelligent wheelchairs: Collision avoidance and navigation assistance for older adults with cognitive impairment. In *Proc. Workshop on Intelligent Systems for Assisted Cognition*. Rochester, NY. 632
- Waldinger, R. (1977). Achieving several goals simultaneously. In E. Elcock and D. Michie (Eds.), *Machine Intelligence 8: Machine Representations of Knowledge*, pp. 94–136. Ellis Horwood, Chichester, England. 367
- Walsh, T. (2007). Representing and reasoning with preferences. *AI Magazine*, 28(4): 59–69. 413

- Warren, D.H.D. and Pereira, F.C.N. (1982). An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3-4): 110-122. 8
- Webber, B.L. and Nilsson, N.J. (Eds.) (1981). *Readings in Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA. 41, 642
- Weiss, G. (Ed.) (1999). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA. 449
- Weiss, S. and Kulikowski, C. (1991). *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann, San Mateo, CA. 341
- Weld, D. (1999). Recent advances in AI planning. *AI Magazine*, 20(2). 367
- Weld, D.S. (1992). Qualitative physics: Albatross or eagle? *Computational Intelligence*, 8(2): 175-186. Introduction to special issue on the future of qualitative physics. 66
- Weld, D.S. (1994). An introduction to least commitment planning. *AI Magazine*, 15(4): 27-61. 367
- Weld, D. and de Kleer, J. (Eds.) (1990). *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, San Mateo, CA. 66
- Whitley, D. (2001). An overview of evolutionary algorithms. *Journal of Information and Software Technology*, 43: 817-831. 152
- Wilkins, D.E. (1988). *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, San Mateo, CA. 367
- Winograd, T. (1990). Thinking machines: Can there be? Are we? In D. Partridge and Y. Wilks (Eds.), *The Foundations of Artificial Intelligence: A Sourcebook*, pp. 167-189. Cambridge University Press, Cambridge, England. 41
- Winograd, T. (1972). *Understanding Natural Language*. Academic Press, New York. 8
- Woods, W.A. (2007). Meaning and links. *AI Magazine*, 28(4): 71-92. 591
- Wooldridge, M. (2002). *An Introduction to Multiagent Systems*. John Wiley and Sons, Chichester, England. 449
- Yang, Q. (1997). *Intelligent Planning: A Decomposition and Abstraction-Based Approach*. Springer-Verlag, New York. 367
- Yang, S., and Mackworth, A.K. (2007). Hierarchical shortest pathfinding applied to route-planning for wheelchair users. In *Proc. Canadian Conf. on Artificial Intelligence, AI-2007*. Montreal, PQ. 632
- Zhang, N.L. and Poole, D. (1994). A simple approach to Bayesian network computations. In *Proc. of the Tenth Canadian Conference on Artificial Intelligence*, pp. 171-178. 274
- Zhang, N.L. (2004). Hierarchical latent class models for cluster analysis. *Journal of Machine Learning Research*, 5(6): 697-723. 341
- Zhang, Y. and Mackworth, A.K. (1995). Constraint nets: A semantic model for hybrid dynamic systems. *Theoretical Computer Science*, 138: 211-239. 66
- Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3): 73-83. 41

索引

索引中的页码为英文原版书的页码,与书中页边标注的页码一致。

$=$ (equals)(相等), 532
 A^* search(A^* 搜索), 89
 \wedge (and)(合取), 163
 \Leftarrow (base-level if)(基级条件), 581
 \Leftarrow (if)(当), 163, 496
 \models (entails)(逻辑蕴含), 160, 499
 \models (true in)(在……中为真), 222
 \neq (not equal to)(不等于), 535
 ϕ (denotation of terms)(表示项), 496, 514
 π (denotation of predicate symbols)(表示谓词符号), 496
 π (meaning of atoms)(原子的含义), 159
 \mapsto (rewritten as)(重写为), 521
 \vdash (derive)(导出), 167
 \vee (or)(析取), 585
 $\%$ (comment)(注释), 496
 $\&$ (base-level and)(基级与), 581

A

abduction(溯因推理), 199, 200
abductive diagnosis(溯因诊断), 201
abilities(能力), 11
absolute error(绝对误差), 290
absorbing state(吸收态), 399
abstractions(抽象), 15
achievement goal(实现的目标), 25, 355
action(动作), 45, 350
action constraints(动作约束), 361
action features(动作特征), 360
action function(动作函数), 74
action profile(动作组合), 425, 435
activation function(激活函数), 306
active learning(主动学习), 285
active sensor(主动传感器), 64
acts(动作), 4
actuator(执行器), 44, 45
acyclic(无环的), 184, 195

acyclic directed graph(无环有向图), 75
add list(添加表), 354
additive independence(累加独立性), 378
additive utility(累加效用), 378
admissibility(可采纳性), 91
aerodynamics(航空动力学), 10
Agent(智能体), 4, 10, 43, 45
 embedded(嵌入式), 59
 purposive(有目标的), 44
Agent system(Agent 系统), 45
Agent system model(Agent 系统模型), 59
AI(人工智能), 3
algebra(代数), 222
algebraic variable(代数变量), 113
Allais Paradox(Allais 悖论), 380
alpha-beta (α - β) pruning(α - β 剪枝), 431
alternative(抉择), 615
analysis(分析), 4
Analytical Engine(分析机), 7
ancestor(祖先), 184
annealing schedule(退火计划), 138
answer(解答, 答案), 166, 171, 187, 504
answer clause(答案子句), 171
answer extraction(解答抽取), 504, 509
anytime algorithm(任意时间算法), 26
application of substitution(替换的应用), 506
approximately correct(近似正确), 332
approximately optimal(近似最优), 14
approximately optimal solution(近似最优解), 14
arc(弧), 75
arc consistent(弧一致), 121
argument(变元), 197, 494
Aristotelian definition(亚里士多德的定义), 567
array(数组), 634
Arrow's impossibility theorem(阿罗不可能定理), 446
artificial intelligence(人工智能), 3

ask(询问), 161, 166
ask-the-user(询问用户), 175, 576
askable(可询问的), 175
assertional knowledge base(断言知识库), 568
assignment space(赋值空间), 118
assumable(假说), 187, 200
assumption-based truth maintenance system(基于假设的真值维持系统), 207
asymptotic complexity(渐进复杂度), 83
asynchronous value iteration(异步值迭代), 406
ATMS(基于假设的真值维持系统), 207
atom(原子), 158, 163
atomic clause(原子子句), 163, 496
atomic proposition(原子命题), 158, 163
atomic symbol(原子符号), 494
attribute(特性), 552
auction(拍卖), 448
autonomous delivery robot(自治传送机器人), 30
average reward(平均回报), 402
axiom(公理), 160, 161, 501
axiom schema(公理模式), 533
axiomatizing(公理化), 161, 501
axioms(公理)
 of probability(概率), 224
 of rationality(理性), 373

B

back-propagation learning(反向传播学习), 317
background knowledge(背景知识), 17, 607
background variables(背景变量), 206
backtracking(回溯), 80, 120
backward induction(逆向归纳法), 430
bagging(bagging 算法), 319
base language(基语言), 580
base level(基本级), 580
base-level algorithms(基准算法), 319
Basic Formal Ontology(基本形式化本体), 573
Bayes' rule(贝叶斯规则), 229
Bayesian classifier(贝叶斯分类器), 309
Bayesian learning(贝叶斯学习), 334
Bayesian network(贝叶斯网络), 参见 belief network
Bayesian probability(贝叶斯概率), 220
beam search(束搜索), 141

belief(信念), 49
belief monitoring(信念监听), 271
belief network(信念网络), 235, 236, 458
 causality(因果关系), 241
 inference problem(推理问题), 252
belief state(信念状态), 48, 61
belief state transition function(信念状态转换函数), 49
best response(最佳响应), 436
best-first search(最佳优先搜索), 88
beta distribution(beta 分布), 337
BFO(基本形式化本体), 573
bias(偏置), 286, 321, 331
bias-free(无偏置), 331
bidirectional search(双向搜索), 101
binary constraint(二元约束), 116
binary feature(二元特征), 112
biology(生物学), 6
bit(位), 231
blame attribution problem(责任归属问题), 465
body(主体), 44
 Agent(Agent), 45
 rule(规则), 163, 496
Boltzmann distribution(波尔兹曼分布), 142, 473
Boolean property(布尔属性), 553
Boolean variable(布尔变量), 113, 126, 158
boosting(boosting 算法), 320
bottom-up proof procedure(自底向上的证明过程), 167
boundary of a process(过程的边界), 576
boundary of an object(对象的边界), 575
bounded rationality(有限理性), 26
branch-and-bound(分支定界), 98
branching factor(分支因子), 77
breadth-first search(宽度优先搜索), 84

C

candidate elimination algorithm(候选删除算法), 329
canonical representation(标准表示), 534
cardinal(基数), 14
cardinal preference(基数偏好), 25
case analysis(实例分析), 125
case-based reasoning(基于案例的推理), 324
causal(有因果联系的), 48

- causal link(因果关系), 364
- causal mechanism(因果机制), 206
- causal model(因果模型), 204, 206
- causal network(因果网络), 241
- causal rule(因果规则), 353, 601
- causal transduction(因果转换), 48
- causality(因果性), 204, 241
- central limit theorem(中心极限定理), 223
- chain rule(链式法则), 227
- chance node(机会节点), 384
- characteristic function(特征函数), 559
- children(子代), 142
- choice space(选择空间), 615
- choose(挑选), 170
- Church-Turing thesis(Church-Turing 理论), 7
- clarity principle(明晰性原则), 114, 241
- Clark normal form(Clark 范式), 538
- Clark's completion(Clark 完备化), 194, 538
- class(类), 298, 309, 451, 559, 568
- classification(分类), 284, 298
- classification tree(分类树), 298
- clause(子句), 126
 - definite(限定), 163, 494, 496
 - Horn, 185
- closed list(closed 表), 94
- closed-world assumption(封闭世界假设), 193
- cluster(簇), 451
- clustering(聚类), 451
- cognitive science(认知科学), 10
- command(指令), 45
- command function(指令函数), 49
- command trace(指令轨迹), 46
- commonsense reasoning(常识推理), 13
- competitive(竞争的), 424
- complements(替换), 381
- complete(完备的), 131, 167
 - bottom-up derivation(自底向上推导), 169
- complete knowledge assumption(完备知识假设), 193
- completeness(完备性)
 - of preferences(偏好), 373
- complex preference(复杂偏好), 25
- complexity(复杂度), 83
- compositional measure of belief(信念的组合度量), 226
- compound proposition(复合命题), 158
- computational(计算), 4
- computational learning theory(计算学习理论), 332
- computational limits dimension(计算限制维度), 26
- computational linguistics(计算语言学), 520
- concept(概念), 572
- conceptualization(概念化), 494, 563
- conditional effect(有条件的影响), 355
- conditional(条件的)
 - expected value(期望), 231
- conditional probability(条件概率), 225
- conditional probability distribution(条件概率分布), 227, 297
- conditionally independent(条件独立), 233
- Condorcet paradox(Condorcet 悖论), 445
- conflict(冲突), 132, 187
- conjunction(合取), 158
- consequence set(结论集合), 167
- consistency-based diagnosis(基于一致性的诊断), 187, 189
- consistent(相容的), 328, 358, 608
- constant(常量), 494
- constrained optimization problem(有约束最优化问题), 144
- constraint(约束), 115
- constraint network(约束网络), 121
- constraint optimization problem(约束优化问题), 145
- constraint satisfaction problem(约束满足问题), 117
- context-free grammar(上下文无关文法), 521
- context-specific independence(上下文特定的独立性), 250
- contingent attribute(或有特性), 562
- continuant(持存体), 573
- continuous variable(连续变量), 113
- controller(控制器), 45, 48, 426
- cooperate(协作), 444
- cooperative(协作的), 424
- cooperative system(协作系统), 196
- coordinate(协调(动词)), 444
- coordination(协调(名词)), 437
- cost(代价), 76
- credit-assignment problem(信度分配问题), 285
- cross validation(交叉验证), 324, 325

crossover(交叉), 142
 CSP, 参见 constraint satisfaction problem
 culture(文化), 6
 cumulative probability distribution(累积概率分布), 257
 cumulative reward(累积回报), 401
 cyc(cyc), 564
 cycle(环), 75
 cycle check(环检查), 93
 cyclic(有环的), 184

D

DAG(有向无环图), 75
 Datalog, 494
 datatype property(数据类型属性), 568
 DBN, 参见 dynamic belief network
 DCG, 参见 definite clause grammar
 dead reckoning(航位推算), 58
 debugging(调试), 179
 incorrect answers(不正确解答), 180
 infinite loops(死循环), 184
 missing answers(缺失的解答), 182
 decision(决策)
 sequential(序贯), 386
 single(单一), 384
 decision function(决策函数), 391
 decision network(决策网络), 384, 387, 428
 decision node(决策节点), 384
 decision tree(决策树), 298, 382, 426
 learning(学习), 298, 321
 decision tree learning(决策树学习), 232
 decision variable(决策变量), 382
 decision-theoretic planning(决策理论规划), 409
 deduction(演绎), 167, 200
 Deep Space One(深度空间一号), 212
 default(默认值), 176, 196
 definite clause(确定子句), 163, 494, 496
 definite clause grammar (DCG)(确定子句文法), 522
 definite clause resolution(确定子句归结), 169
 delay(推迟), 536, 590
 delete list(删除表), 354
 delivery robot(传送机器人), 30
 DENDRAL(DENDRAL 专家系统), 9
 denote(指示), 496
 dense(稠密), 46
 dependent continuant(从属持存体), 573, 575
 depth-bounded meta-interpreter(深度有界元解释器), 587
 depth-first search(深度优先搜索), 80
 derivation(推导), 167, 171, 510
 derived(导出), 167, 204, 354, 600
 knowledge(知识), 558
 description logic(描述逻辑), 568
 design(设计), 4, 202
 design space(设计空间), 19
 design time reasoning(设计时推理), 17
 desire(愿望), 49
 deterministic(确定的), 24
 diagnosis(诊断), 201
 abductive(溯因), 201
 consistency-based(基于一致性的), 187
 diagnostic assistant(诊断助手), 30, 33
 dictator(专断 Agent), 447
 difference list(差异列表), 522
 differentia(区别特征), 567
 dimension(维度)
 computational limits(计算限制), 26
 effect uncertainty(影响不确定), 24
 learning(学习), 26
 modularity(模块性), 19
 number of Agents(Agent 数量), 25
 planning horizon(规划时域), 22
 preference(偏好), 24
 representation scheme(表示方案), 20
 sensing uncertainty(感知不确定性), 23
 directed acyclic graph(有向无环图), 75
 directed graph(有向图), 75
 Dirichlet distribution(Dirichlet 分布), 337
 discount factor(折扣因子), 402
 discounted reward(折扣回报), 402
 discrete(离散的), 46
 discrete variable(离散变量), 113
 disjunction(析取), 158, 185, 585
 disjunctive normal form(析取范式), 189
 disposition(处置), 575
 distribution(分布)
 normal(正态), 223
 do(do(执行)), 598
 domain(域), 112, 113, 496, 552, 633
 domain consistent(域一致), 121

domain expert(领域专家), 63
 domain ontology(领域本体), 571
 domain splitting(域分割), 125
 dominant strategy(支配策略), 446
 dominates(支配), 439
 don't-care non-determinism(不介意不确定性), 170
 don't-know non-determinism(不知道不确定性), 170
 dot product(点积), 271
 DPLL, 127
 dynamic(动态), 599
 dynamic belief network(动态信念网络), 272
 dynamic decision network(动态决策网络), 409
 dynamic programming(动态规划), 103

E

economically efficient(经济有效), 446
 effect(影响), 350, 354
 effect constraints(影响约束), 361
 effect uncertainty dimension(影响不确定维度), 24
 effectively computable(能行可计算), 7
 effector(效应器), 44
 efficient(高效的), 446
 eligibility trace(资格迹), 479
 elimination ordering(消除排序), 252
 EM(EM, 期望最大化), 452, 455, 460
 embedded Agent(嵌入式 Agent), 59
 embodied(物化的), 44
 empirical frequency(经验频率), 295
 endogenous variables(内因变量), 206
 engineering goal(工程目标), 4
 ensemble learning(集成学习), 319
 entails(逻辑蕴含), 160
 entity(实体), 573
 entropy(熵), 232, 292
 environment(环境), 10
 epistemological(认识论的), 221
 epistemology(认识论), 9
 equal(相等), 532
 error(误差), 288
 absolute(绝对), 290
 sum squares(平方和), 290
 worst case(最坏情况下), 291
 error of hypothesis(假设的误差), 332
 Euclidean distance(欧氏距离), 95, 325
 evaluation(评估), 145

evaluation function(评价函数), 132, 433
 event calculus(事件演算), 604
 evidence(证据), 225
 evidential model(证据模型), 205
 evolutionary algorithm(演化算法), 466
 exclusive-or(异或), 308
 existence uncertainty(存在的不确定性), 619
 existentially quantified variable(存在量化的变量), 500
 exogenous variables(外因变量), 206
 expanding(扩展), 78
 expectation maximization (EM)(期望最大化), 452
 expected monetary value(期望货币价值), 376
 expected utility(期望效用), 386, 392
 expected value(期望值), 230
 of utility of a policy(策略的效用的), 392
 experience(经历), 468
 expert opinion(专家观点), 297
 expert system(专家系统), 9, 10, 61
 explained away(解释消除), 201, 243, 245
 explanation(解释), 201
 exploit(利用), 472
 explore(探索), 472
 expression(表达式), 496
 extensional(外延的), 559
 extensionally(外延地), 116
 extensive form(扩展形式), 426
 external knowledge source(外部知识源), 65
 extrapolation(外推), 286

F

$f(p)$, 89
 factor(因子), 249, 634
 factored finite state machine(因素化有限状态机), 50
 factored representation(因素化表达), 50
 factorization(因式分解), 236
 fail(失败), 170
 failing naturally(正常失败), 96
 failing unnaturally(非正常失败), 96
 failure(失败), 197
 fair(公平的), 170, 518
 fairness(公平), 19
 false(假), 159, 498

false-negative error(错否定错误), 14, 182, 292
 false-positive error(错肯定错误), 14, 180, 292, 293
 fault(缺陷), 187
 feature engineering(特征工程), 484
 feature-based representation of actions(基于特征的动作表示法), 353
 features(特征), 21, 112
 feed-forward neural network(前馈神经网络), 316
 fiat part of an object(对象的特殊部分), 575
 fiat part of process(过程的特殊部分), 576
 filler(填充物), 556
 filtering(过滤), 58, 267, 271
 finite failure(有限的失败), 199
 finite horizon(有限时域), 23
 finite state controller(有限状态控制器), 50
 finite state machine(有限状态机), 50
 first-order predicate calculus(一阶谓词演算), 517
 fixed point(不动点), 169
 flat(扁平的), 20
 floundering goal(紊乱的目标), 541
 fluent(流), 599
 flying machines(飞行器), 9
 for all(对于所有的), 500
 forward chaining(正向链接), 167
 forward planner(前向规划器), 356
 frame(框架), 556
 frame problem(框架问题), 618
 frame rule(框架规则), 353, 601
 framing effect(框架效应), 380
 frequentist(频率论者的), 220
 fringe(边缘), 77
 frontier(前沿), 77
 fully observable(完全可观察), 23
 fully observable dynamic decision network(完全可观察动态决策网络), 411
 fully observable Markov decision process(完全可观察马尔可夫决策过程), 401
 function(功能), 575, 633
 symbol(符号), 514
 fuzzy terms(模糊项), 52

G

gambling(赌博), 220
 game tree(博弈树), 426

Gaussian distribution(高斯分布), 223
 general boundary(一般边界), 329
 generalized additive independence(广义累加独立性), 381
 generalize(泛化), 285
 generalized answer clause(一般解答子句), 509
 generalized arc consistency algorithm(广义弧一致算法), 122
 generate and test(生成并测试), 118, 146
 genetic algorithms(遗传算法), 142
 genus(种类), 567
 Gibbs distribution(Gibbs 分布), 142, 473
 Gini index(Gini 指数), 302, 345
 global minimum(全局最小值), 149
 goal(目标), 11, 25, 49, 171, 356
 node(节点), 75
 goal constraints(目标约束), 361
 gradient descent(梯度下降), 149, 304
 grammar(文法), 521
 context-free(上下文无关), 521
 definite clause(限定子句), 522
 graph(图), 75
 greedy(贪婪), 23, 609
 greedy ascent(贪婪上升), 132
 greedy descent(贪婪下降), 132
 ground(基), 496
 ground instance(基实例), 506, 507
 ground representation(基础表示), 580

H

$h(n)$, 87
 hard clustering(硬聚类), 452
 hard constraint(硬性约束), 111, 115
 head(头), 163, 496
 help system(帮助系统), 246, 312
 Herbrand interpretation(Herbrand 解释), 508
 heuristic(启发式)
 function(函数), 87
 knowledge(知识), 72
 search(搜索), 87
 heuristic depth-first search(启发式深度优先搜索), 88
 hidden Markov model (HMM)(隐马尔可夫模型), 267
 hidden variable(隐藏变量), 244, 460

hierarchical(层次结构的), 20
 hierarchical Bayesian model(层次贝叶斯模型), 338
 hierarchical control(分层控制), 50
 hill climbing(爬山算法), 132
 history(历史), 10, 48, 468
 HMM, 参见 hidden Markov model
 Hoeffding's inequality(Hoeffding 不等式), 259
 horizon(时域), 361
 Horn clause(Horn 子句), 185
 how(如何), 17
 how question(“如何”问题), 177
 human-computer interaction (HCI)(人机交互), 578
 hyperparameters(超参数), 339
 hypothesis(假设), 288
 hypothesis space(假设空间), 288, 328

I

identity uncertainty(身份不确定性), 619
 imperfect information game(不完全信息博弈), 428
 implication(蕴含), 158
 incoming arc(入弧), 75
 inconsistent(不一致), 186
 incorrect answer(不正确答案), 180
 incremental gradient descent(增量梯度下降), 305
 indefinite horizon(无限定时域), 23, 399
 independent(独立于), 233
 independent and identically distributed (i. i. d.)(独立同分布), 335
 independent choice logic (ICL)(独立选择逻辑), 615
 independent continuant(独立持存体), 573, 575
 indicator variable(指示变量), 141, 290
 indifferent(无差别), 373
 individual(个体), 22, 141, 492, 496, 568
 individual-property-value(个体-属性-值), 552
 induction(归纳), 200, 284
 inductive logic programming(归纳逻辑编程), 606
 inference(推理), 167
 infinite horizon(无穷时域), 23, 399
 influence diagram(影响图), 387
 information content(信息内容), 232
 information gain(信息增益), 232, 302

information set(信息集合), 428
 information theory(信息理论), 231, 323
 init(init(初始情景)), 598
 initial-state constraints(初始状态约束), 361
 input features(输入特征), 288
 insects(昆虫), 18
 instance(实例), 504, 506
 ground(基), 507
 instance space(实例空间), 328
 insurance(保险), 377
 integrity constraint(完整性约束), 185
 intelligent tutoring system(智能指导系统), 35
 intended interpretation(预期解释), 161, 501
 intensional(内涵的), 559
 intensionally(内涵地), 116
 intention(意图), 49
 interpolation(插值), 286
 interpretation(解释), 159, 496
 intersection(交), 636
 intervention(干预), 204, 206, 241
 involve(涉及), 115
 island(孤岛), 102
 island-driven search(孤岛驱动搜索), 101
 iterative best improvement(迭代最佳改进), 132
 iterative deepening(迭代深化), 95
 A^* (A^* 算法), 98

J

Java(一种面向对象的编程语言), 495, 558
 join(连接), 636
 joint probability distribution(联合概率分布), 236, 252

K

k -fold cross validation(k -折交叉验证), 324
 k -means algorithm(k -均值算法), 452
 kd -tree(kd -树), 326
 kernel functions(核函数), 314
 knowledge(知识), 12, 60
 knowledge base(知识库), 12, 17, 60, 160, 163, 494, 496
 knowledge engineers(知识工程师), 63
 knowledge is given(知识是给定的), 26
 knowledge is learned(知识是学习到的), 26
 knowledge level(知识层), 16, 176

knowledge-based system(基于知识的系统), 60
knowledge-level debugging(知识层的调试), 180

L

landmarks(标志), 52
language(语言), 521
 natural(自然), 520
language bias(语言偏置), 331
latent tree model(潜在树模型), 314
latent variable(潜在变量), 244, 309, 460
learning(学习), 61, 283-347, 441-445, 451-488, 606-611
 Bayesian(贝叶斯), 334
 bias(偏置), 331
 case-based(基于案例的), 324
 decision trees(决策树), 232, 298
 decision tree(决策树), 321
 multiAgent(多 Agent), 441
neural network(神经网络), 315
PAC, 332, 340
relational(关系的), 606
supervised(有监督的), 284
to coordinate(协调), 441
unsupervised(无监督的), 285, 451
version space(变型空间), 329
learning bias(学习偏置), 321
learning dimension(学习维度), 26
learning rate(学习率), 304
leave-one-out cross-validation(留一法交叉验证), 325
leaves(叶节点), 75
level of abstraction(抽象的层次), 15
life-long learning(终生学习), 6
likelihood(可能性), 229
likelihood of the data(数据的似然性), 292
linear function(线性函数), 304
linear regression(线性回归), 304, 484
linearly separable(线性可分), 308
Linnaean taxonomy(林奈分类), 567
lists(列表), 516
literal(文字), 126, 194
liveness(活跃性), 18
local minimum(局部极小), 149
local optimum(局部最优), 132
local search(局部搜索), 130, 131, 609

localization(定位), 268
logic(逻辑), 495
logic program(逻辑程序), 514
logic programming(逻辑编程), 207
logical consequence(逻辑结论), 160, 499
logical variable(逻辑变量), 494
logically follows(逻辑遵循), 160
logistic function(logistic 函数), 307
long-term memory(长期记忆), 61
loop check(循环检查), 93
lottery(抽奖法), 373
lowest-cost-first search(最低代价优先搜索), 86

M

M features(M 特征), 360
M system(M 系统), 51
machine learning(机器学习), 参见 learning
maintenance goal(维持性目标), 25, 355
MAP model(MAP 模型), 321
mapping(映射), 633
margin(间隔), 314
Markov assumption(马尔可夫假设), 266
Markov blanket(马尔可夫毯), 235
Markov chain(马尔可夫链), 266
Markov decision process(马尔可夫决策过程), 399, 463
matrix multiplication(矩阵乘法), 271
maximum a posteriori probability(最大化后验概率), 321
maximum entropy(最大熵), 233, 234
maximum likelihood model(最大似然模型), 292, 321
maximum-likelihood estimate(最大似然估计), 295
MDL principle(MDL 准则), 323
MDP, 参见 Markov decision process
measure(度量), 221
mechanism(机制), 424, 446
mechanism design(机制设计), 446
meta-interpreter(元解释器), 579, 580, 582
 ask the user(询问用户), 589
 build proof tree(构建证明树), 587
 built-in predicates(内置谓词), 586
 delayed goals(推迟的目标), 590
 depth-bounded(深度有限), 587
 disjunction(析取), 586

vanilla(普通的), 583
 meta-level(元级), 580
 metalanguage(元语言), 580
 MGU(最广合一算子), 507
 minimal(最小)
 conflict(冲突), 187
 model(模型), 169
 minimal diagnosis(最小诊断), 189
 minimal explanation(最小解释), 201
 minimal model(最小模型), 509
 minimax(极大极小), 430
 minimum description length (MDL)(最小描述长度), 323
 minimum fixed point(最小不动点), 169
 missing at random(随机缺失), 461
 model(模型), 15, 57, 116, 160
 modular(模块化的), 20
 modularity(模块性), 19
 modularity dimension(模块性维度), 20
 modus ponens(假言推理), 167
 money pump(货币泵), 374
 monitoring(监督), 267, 271
 monotone restriction(单调限制), 95
 monotonic(单调的), 196
 more general hypothesis(更一般的假设), 329
 more specific hypothesis(更明确的假设), 329
 most general unifier(最广合一算子), 507, 511
 multiAgent decision network(多 Agent 决策网络), 428
 multiple Agent(多 Agent), 25
 multiple-path pruning(多路径剪枝), 93, 94
 mutex constraint(互斥约束), 361
 MYCIN, 9
 myopic(短视的), 23
 myopically optimal(目前看来是最优), 301

N

N3, 555
 naive Bayesian classifier(朴素贝叶斯分类器), 246, 310, 455
 Nash equilibrium(纳什均衡), 436
 natural kind(自然类), 309, 313, 559
 natural language processing(自然语言处理), 520
 nature(自然状态), 44, 424
 nearest neighbors(最近邻), 325

negation(否定), 158, 185
 negation as failure(否定即失败), 194, 199, 537
 negative examples(负类样本), 607
 neighbor(邻居), 75, 131
 neural network(神经网络), 315
 no, 166
 no-forgetting Agent(无遗忘 Agent), 388
 no-forgetting decision network(无遗忘决策网络), 388
 node(节点), 75
 noisy(噪声), 267
 noisy or(噪声或), 250
 non-deterministic(不确定的), 170
 non-deterministic procedure(不确定性过程), 170
 non-ground representation(非基表示), 580
 non-monotonic(非单调的), 196
 non-parametric distribution(非参数化分布), 223
 non-planning(非规划), 23
 non-serial dynamic programming(非序列动态规划), 151
 non-terminal symbol(非终结符号), 521
 nonlinear planning(非线性规划), 364
 normal distribution(正态分布), 223
 normal-form game(标准式博弈), 425
 NP-complete(NP 完全), 170
 NP-hard(NP 难), 170
 number of Agents dimension (Agent 数量维度), 25
 number uncertainty(数量不确定性), 619

O

object(对象), 552, 575
 object aggregate(对象聚合), 575
 object language(对象语言), 580
 object property(对象属性), 568
 object-oriented languages(面向对象语言), 495
 object-oriented programming (OOP) languages(面向对象编程语言), 558
 objective function(目标函数), 144
 observation(观察), 11, 17, 174, 225
 occurrent(偶然体), 573, 575
 occurs check(出现检查), 518
 Ockham's razor(奥卡姆剃刀原理), 287
 off policy(离策略), 470
 off-policy learner(离策略学习器), 475

offline(离线), 61
 offline computation(离线计算), 17
 offline learning(离线学习), 285
 omniscient agent(全知 Agent), 114
 on-policy learner(在策略学习器), 475
 one-point crossover(单点交叉), 142
 online(在线), 60, 64
 online computation(在线计算), 17
 online learning(在线学习), 285
 ontological(本体论的), 221
 ontology(本体), 61, 161, 175, 549, 563
 OOP, 参见 object-oriented programming
 open-world assumption(开放世界假设), 193
 optimal(最优的), 74
 optimal algorithm(最优算法), 105
 optimal policy(最优策略), 386, 392, 403
 optimal solution(最优解), 13, 76
 optimality criterion(最优性标准), 144
 optimization problem(最优化问题), 144
 oracle(预言), 170
 orders of magnitude reasoning(量级推理), 52
 ordinal(与顺序有关的), 13
 ordinal preference(与顺序有关的偏好), 25
 organizations(组织), 6
 outcome(结果), 373, 382, 425
 outgoing arc(出弧), 75
 overfitting(过拟合), 303
 OWL(Web 本体语言), 564, 568

P

PAC learning(PAC 学习), 332, 340
 pair(对), 633
 parametric distribution(参数分布), 223
 parametrized random variable(参数化随机变量), 613
 paramodulation(参数化调整), 534
 parents(父节点), 235
 partial(部分的), 267
 partial evaluation(部分评价), 590
 partial-order planning(偏序规划), 363
 partially observable(部分可观察), 23
 partially observable game(部分可观察博弈), 428
 partially observable Markov decision process(部分可观察马尔可夫决策过程), 401, 411
 particle(粒子), 264
 particle filtering(粒子滤波), 264, 265
 passive sensor(被动传感器), 64
 past experience(过去的经验), 11
 path(路径), 75
 consistency(一致性), 125
 payoff matrix(支付矩阵), 426
 percept(感知), 45
 percept stream(感知流), 46
 percept trace(感知轨迹), 46
 perception(感知), 58
 perceptron(感知器), 306
 perfect information(完全信息), 430
 perfect information game(完全信息博弈), 426
 perfect rationality(完全理性), 26
 philosophy(哲学), 9
 physical symbol system(物理符号系统), 15
 physical symbol system hypothesis(物理符号系统假设), 15, 319
 pixels(像素), 45
 plan(规划), 356
 planner(规划器), 356
 planning(规划), 356-370, 604
 as a CSP(作为一个 CSP), 360
 forward(前向), 356
 partial-order(偏序), 363
 regression(回归), 357
 planning horizon(规划时域), 22
 planning horizon dimension(规划时域维度), 23
 plate model(平板模型), 338, 613
 point estimate(点估计), 288
 policy(策略), 103, 386, 390
 policy iteration(策略迭代), 407
 policy search(策略搜索), 466
 POMDP, 参见 partially observable Markov decision process
 population(种群), 141, 264
 positive examples(正类样本), 607
 possible(可接受的), 358
 possible world(可能世界), 113, 115, 382, 391, 615
 posterior probability(后验概率), 225
 pragmatics(语用), 521
 precision(精确度), 293
 precision-recall curve(准确度-召回率曲线), 293
 precondition(先决条件), 350, 353, 354, 601
 precondition constraints(先决条件约束), 361

predicate symbol(谓词符号), 494
 preference(偏好), 144
 preference bias(偏好偏置), 331
 preference dimension(偏好维度), 25
 preference elicitation(偏好引出), 379
 primitive(原始), 204, 354, 566, 600
 knowledge(知识), 558
 prior count(先验计数), 296
 prior knowledge(先验知识), 10
 prior probability(先验概率), 225
 prisoner's dilemma(囚徒困境), 437
 probabilistic independence(概率独立性), 233
 probabilistic inference(概率推理), 248
 probabilistic relational model (PRM)(概率关系模型), 612
 probability(概率), 219-275, 295
 axioms(公理), 224
 conditional(条件), 225
 posterior(后验), 225
 prior(先验), 225
 probability density function(概率密度函数), 223
 probability distribution(概率分布), 222
 probability mass(概率质量), 262
 probability measure(概率度量), 221
 probable solution(可能解), 14
 probably approximately correct(可能近似正确), 259, 332
 probably approximately correct learning(可能近似正确学习), 332
 process(过程), 23, 576
 process aggregate(过程聚合), 576
 processual context(过程性上下文), 576
 processual entity(过程性实体), 576
 projection(投影), 636
 Prolog, 9, 494, 496
 proof(证明), 167
 bottom-up(自底向上), 167
 top-down(自顶向下), 169
 proof procedure(证明过程), 167
 proof tree(证明树), 179
prop, 552
 property(属性), 552, 568
 property inheritance(属性继承), 561
 proposal distribution(提议分布), 260, 261
 proposition(命题), 21, 158
 propositional calculus(命题演算), 158

propositional definite clause resolution(命题确定子句解析), 169
 propositional definite clauses(命题确定子句), 163
 propositional satisfiability(命题可满足性), 126
 prospect theory(前景理论), 380
 proved(证明), 167
 pseudocount(伪计数), 296
 psychology(心理学), 9
 punishment(惩罚), 399
 pure strategy(纯策略), 435
 pure symbol(纯符号), 127
 purposive agent(有目标的 Agent), 44

Q

Q^* , 404
 Q-learning(Q-学习), 469
 Q-value(Q-值), 404, 469
 $Q\pi$, 403
 qualitative derivatives(定性导出), 52
 qualitative reasoning(定性推理), 52
 quality(特征), 575
 quantitative reasoning(定量推理), 52
 query(查询), 166, 494, 496
 querying the user(询问用户), 175

R

random initialization(随机初始化), 131
 random restart(随机重启), 131
 random sampling(随机采样), 132
 random variable(随机变量), 221
 random walk(随机游走), 132
 random worlds(随机世界), 233, 234
 range(范围), 552, 633
 rational(理性), 376
 rational agent(理性 Agent), 373
 RDF, 555, 559, 564
 RDF Schema, 559
 RDF-S, 564
 rdf: type, 559
 rdfs: subClassOf, 559
 reachable(可达的), 599
 realizable entity(可实现实体), 575
 reasoning(推理), 17
 recall(召回), 293
 recognition(确认), 201

record linkage(记录关联), 619
 recursively enumerable(递归可数), 517
 reflection(慎思), 579
 regression(回归), 284, 304
 regression planning(回归规划), 357, 358
 regression tree(回归树), 314
 reify(具体化), 552
 reinforcement learning(增强学习), 285, 463
 rejection sampling(舍选采样), 259
 relation(关系), 22, 492, 633, 635
 relational algebra(关系代数), 635
 relational database(关系数据库), 635
 relational descriptions(关系描述), 22
 relational probability model(关系概率模型), 612
 relational uncertainty(关系的不确定性), 619
 representation(表示), 12
 representation scheme(表示方案), 12
 representation scheme dimension(表示方案维度), 20, 22
 resampling(重采样), 264
 resolution(归结), 169, 171
 SLD, 169, 510
 resolvent(分解), 171
 resource(资源), 555, 564
 Resource Description Framework(资源描述框架), 555, 564
 restriction bias(限制偏置), 331
 return(返回值), 469
 revelation principle(显示原理), 447
 reward(回报), 399
 rewrite rule(重写规则), 521, 534
 risk averse(风险规避), 377
 robot(机器人), 10, 30, 44
 ROC curve(ROC 曲线), 293
 role(角色), 63, 575
 root(根), 75
 rule(规则), 163, 494, 496
 rule of derivation(推导规则), 167
 run(运行), 427
 run-time distribution(运行时间分布), 138

S

safety goal(安全目标), 18, 355
 sample average(样本均值), 258
 sample complexity(样本复杂度), 333

SARSA(λ), 479
 satisfiable(可满足的), 201
 satisficing(满意), 14
 satisficing solution(满意解), 14
 satisfies(满足), 116
 policy(策略), 391
 satisfy(满足), 115
 scenario(情景), 201
 scheme(方案), 115, 635
 scientific goal(科学目标), 4
 scope(范围), 115, 145, 634, 635
 search(搜索), 71-110
 A^* , 89
 best-first(最佳优先), 88
 bidirectional(双向), 101
 breadth-first(宽度优先), 84
 cycle-checking(环检查), 93
 depth first(深度优先), 80
 dynamic programming(动态规划), 103
 gradient descent(梯度下降), 149
 heuristic(启发式), 87
 island driven(孤岛驱动), 101
 iterative deepening(迭代深化), 95
 local search(局部搜索), 130
 lowest-cost-first(最低代价优先), 86
 search and score(搜索与评分), 462
 search bias(搜索偏置), 331
 second-order logic(二阶逻辑), 517
 second-price auction(第二价格拍卖), 448
 select(选择), 170
 selection(选择), 635
 selector function(选择器函数), 615
 semantic interoperability(语义互操作), 65, 550
 semantic network(语义网), 554
 semantic Web(语义 Web), 555, 564
 semantics(语义), 159, 520
 propositional calculus(命题演算), 159
 variables(变量), 496
 sensing uncertainty dimension(感知不确定维度), 23
 sensor(传感器), 44, 45, 64
 sensor fusion(传感器融合), 270
 separable control problem(可分割的控制问题), 58
 sequential decision problem(序贯决策问题), 386
 sequential prisoner's dilemma(序贯囚徒困境), 438
 set(集合), 633
 set difference(集合差), 636

- short-term memory(短期记忆), 61
 - sigmoid, 307
 - simulated agent(仿真 Agent), 59
 - simulated annealing(模拟退火), 136
 - single agent(单个 Agent), 25
 - single decision(单一决策), 384
 - single-stage decision network(单阶段决策网络), 384
 - site(位置), 575
 - situation(情景), 598
 - situation calculus(情景演算), 598
 - SLD derivation(SLD 推导), 171, 510
 - SLD resolution(SLD 归结), 169, 510
 - slot(槽), 556
 - Smalltalk, 558
 - smoothing(平滑), 267, 271
 - SNLP, 370
 - social preference function(社会偏好函数), 446
 - society(社会), 6
 - soft clustering(软聚类), 452
 - soft constraint(软性约束), 111, 145
 - soft-max(软最大化), 473
 - software agent(软件 Agent), 10
 - software engineer(软件工程师), 63
 - software engineering(软件工程), 13
 - solution(解), 73, 75
 - sound(可靠的), 167
 - spatial region(空间范围), 573, 575
 - spatio-temporal region(时空范围), 576
 - specialization operator(具体化算子), 609
 - specific boundary(特定的边界), 329
 - squashed linear function(扁线性函数), 306
 - stable(稳定的), 453
 - stack(栈), 80
 - stage(阶段), 23
 - start node(起始节点), 75
 - start states(初始状态), 74
 - starvation(饥饿), 170, 518
 - state(状态), 20, 72
 - state constraints(状态约束), 361
 - state space(状态空间), 72
 - state-space graph(状态空间图), 351, 356
 - state-space problem(状态空间问题), 74
 - static(静态), 599
 - stationary(平稳的), 266, 399
 - stationary policy(平稳策略), 403
 - step function(阶跃函数), 306
 - step size(步长), 150, 151
 - stimuli(刺激), 45
 - stochastic(随机的), 24
 - stochastic beam search(随机集束搜索), 142
 - stochastic local search(随机局部搜索), 134
 - stochastic simulation(随机模拟), 256
 - stochastic strategy(随机策略), 435
 - stopping state(停止态), 399
 - strategic form of a game(博弈的策略形式), 425
 - strategically(策略地), 424
 - strategy(策略), 426, 427, 435
 - strategy profile(策略组合), 427, 435
 - strictly dominated(严格被支配的), 147
 - strictly dominates(严格支配), 440
 - strictly preferred(严格偏好), 373
 - STRIPS assumption(STRIPS 假设), 354
 - STRIPS representation(STRIPS 表示), 354
 - structure learning(结构学习), 461
 - subClassOf, 559
 - subgame-perfect equilibrium(子博弈完全均衡), 439
 - subgoal(子目标), 171
 - subject(主题), 552
 - subjective probability(主观概率), 220
 - substitutes(替换), 381
 - substitution(替换), 506
 - sum-of-squares error(平方和误差), 290, 453
 - supervised learning(有监督学习), 284, 288
 - support set(支持集), 435, 439
 - support vector machine (SVM)(支持向量机), 314
 - SVM, 参见 support vector machine
 - symbol(符号), 15, 114
 - symbol level(符号层), 17
 - symbol system(符号系统), 15
 - syntax(句法), 520
 - Datalog, 494
 - propositional definite clauses(命题限定子句), 163
 - synthesis(综合), 4
 - systematicity(系统性), 370
- T
- tabu list(禁忌表), 135
 - target features(目标特征), 288
 - TD error(TD 误差), 467
 - tell(告诉), 161
 - temperature(温度), 473

temporal difference error(时间差分误差), 467
temporal region(时域范围), 575
term(项), 494, 514
terminal symbol(终结符号), 521
terminological knowledge base(术语知识库), 568
test examples(测试样本), 288
theorem(定理), 167
there exists(存在), 500
thing(事物), 573
thinking(思考), 4
threat(威胁), 438
time(时间), 46, 598
tit-for-tat(一报还一报), 438
top-down proof procedure(自顶向下的证明过程), 169
total assignment(全赋值), 115
total reward(总回报), 402
trading agent(交易 Agent), 30, 37
training examples(训练样本), 284, 288, 328
transduction(转换), 48
transitivity of preferences(偏好的传递性), 374
tree(树), 75
tree augmented naive Bayesian (TAN) network(树增强朴素贝叶斯网络), 314
treewidth(树宽), 130, 256
triple(三元组), 552, 555, 606, 633
triple representation(三元组表示), 552
true(真), 159, 498, 633
true-positive rate(真正率), 293
truth maintenance system(真值维持系统), 207
truth value(真值), 159
truthful(真正的), 446
try(尝试), 131
tuple(元组), 115, 633, 635
Turing test(图灵测试), 5
Turtle(一种语言), 555
tutoring system(指导系统), 30
two-step belief network(两步信念网络), 272
type(类型), 559

U

UML, 558
unary constraint(一元约束), 116
unconditionally independent(无条件独立), 235
unification(合一), 511
unifier(合一算子), 507

Uniform Resource Identifier(统一资源标识符), 555, 564, 568
uninformed search strategies(盲目搜索策略), 80
union(并), 636
unique names assumption(唯一名字假设), 535
unit resolution(单元归结), 127
units(单元), 315
universally quantified(全称量化), 498
universally quantified variable(全称量化的变量), 498
unsatisfiable(不可满足的), 186
unstable(不稳定的), 319
unsupervised learning(无监督学习), 285, 451
URI, 参见 Uniform Resource Identifier
useful(令人满意的), 358
user(用户), 64, 114, 175
utility(效用), 14, 376
utility node(效用节点), 385)

V

V^* , 403
 V^* , 404
validation set(验证集), 324
value(值), 401
value of control(控制价值), 398
value of information(信息价值), 397
vanilla meta-interpreter(普通的元解释器), 582, 583
variable(变量), 113
 algebraic(代数的), 113
 Boolean(布尔), 113
 continuous(连续), 113
 decision(决策), 382
 discrete(离散), 113
 existentially quantified(存在量化), 500
 logical(逻辑的), 494
 random(随机), 221
 universally quantified(全称量化), 498
variable assignment(变量指派), 498
variable elimination(变量消除), 386
belief networks(信念网络), 248
CSP, 127
decision networks(决策网络), 392
variational inference(变分推理), 248
VCG mechanism(VCG 机制), 447
VE, 参见 variable elimination

verb(动词), 552
version space(变型空间), 329
virtual body(虚拟体), 50

W

walk(游走), 131
weakest precondition(最弱前提条件), 358
weakly dominated(弱支配), 147
weakly preferred(弱偏好), 373
Web Ontology Language(Web 本体语言), 564
what(哪些), 17
why question(“为什么”问题), 177
whynot question(“为什么不”问题), 177
win or learn fast (WoLF)(赢或快速学习), 445

word(字), 494
world(世界), 10
worst-case error(最坏情况误差), 291
wrapper(封装), 65

X

XML, 564

Y

yes, 166

Z

zero-sum game(零和博弈), 424, 430